NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

INVESTIGATION AND IMPLEMENTATION OF AN ALGORITHM FOR COMPUTING OPTIMAL SEARCH PATHS

by

James F. Caldwell, Jr.

September 1987

Thesis Advisor: James N. Eagle

AD-A185 921

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): 55	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification) Investigation And Implementation Of An Algorithm For Computing Optimal Search Paths

12 PERSONAL AUTHOR(S) CALDWELL, James F., Jr.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	1987 September	88

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Optimal Search Paths, Frank-Wolfe Method, Branch-And-Bound, Search, Optimal Search,

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

A moving target is detected at long range with an initial position given by a probability distribution on a grid of N cells, Also located on the grid is a searcher, constrained by speed, who must find an optimal search path in order to minimize the probability of target survival by time T. A branch-and bound algorithm designed by Professors Eagle and Yee of the Naval Postgraduate School in Monterey, California, is successfully implemented in order to solve this problem. Within the algorithm, the problem is set up as a nonlinear optimization of a convex objective function subject to the flow constraints of an acyclic N x T network. Lower bounds are obtained via the Frank-Wolfe method of solution specialized for acyclic networks. This technique relies on linearization of the objective function to yield a shortest path problem

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: James N. Eagle, Prof.	22b TELEPHONE (Include Area Code): 408-646-2654; 22c OFFICE SYMBOL: 55Er

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Block 19 Abstract Continued

that is solvable by dynamic programming. For each iteration, the lower bound can be found by use of a Taylor first order approximation. Implementation of this algorithm is accomplished by the use of a Fortran program which is run for several test cases. The characteristics of the solution procedure as well as program results are discussed in detail. Finally, some real world applications along with several questions requiring further research are proposed.

Investigation and Implementation of an Algorithm for Computing Optimal Search Paths

by

James F. Caldwell, Jr.
Lieutenant, United States Navy
B.S., United States Naval Academy, 1981

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1987

Author: James F. Caldwell, Jr.

Approved by: James N. Eagle, Thesis Advisor

Richard E. Rosenthal, Second Reader

Peter Purdue, Chairman,
Department of Operations Research

James M. Fremgen,
Acting Dean of Information and Policy Sciences

ABSTRACT

A moving target is detected at long range with an initial position given by a probability distribution on a grid of N cells. Also located on the grid is a searcher, constrained by speed, who must find an optimal search path in order to minimize the probability of target survival by time T. A branch-and-bound algorithm designed by Professors Eagle and Yee of the Naval Postgraduate School in Monterey, California, is successfully implemented in order to solve this problem. Within the algorithm, the problem is set up as a nonlinear optimization of a convex objective function subject to the flow constraints of an acyclic N x T network. Lower bounds are obtained via the Frank-Wolfe method of solution specialized for acyclic networks. This technique relies on linearization of the objective function to yield a shortest path problem that is solvable by dynamic programming. For each iteration, the lower bound can be found by use of a Taylor first order approximation. Implementation of this algorithm is accomplished by the use of a Fortran program which is run for several test cases. The characteristics of the solution procedure as well as program results are discussed in detail. Finally, some real world applications along with several questions requiring further research are proposed.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

Consider a target detected at long range with some position uncertainty, and a searcher, constrained by speed, tasked with closing the range to the target within a specified period of time. Intelligence estimates regarding the target's probable movements are provided to the searcher who must use this information to develop a search path that will bring him close enough to the target for tracking or possibly weapons delivery. This problem may represent an antisubmarine warfare (ASW) search in which a single surface ship attempts to localize a submarine contact. In some instances near-optimal solutions can be obtained based on experience, yet in many other cases the best search path may not be readily apparent. Current search practice dictates a systematic approach to this problem such as sweeping out areas without overlapping until the entire area of uncertainty has been thoroughly swept. Intuitively this procedure seems correct, however this has never been established as the best or even nearly optimal search technique. It is for this reason that the study of optimal search paths is of interest, for in discovering optimal paths for various problems new insights into search theory may be gained.

B. PROBLEM DEFINITION

A grid of N cells as shown in Figure 1.1 is constructed on which a target and a searcher are located. The target's starting position may be represented as a single cell or perhaps by a probability distribution over any number of cells, while the searcher's initial position is specified as one particular cell. The problem proceeds discretely through a series of searches and movements that span a finite time interval of duration T. For each time period, the target moves according to a Markov transition matrix that is defined from prior intelligence known to the searcher. Searcher movements are constrained such that if he currently occupies cell i, in the next time period he may travel only to the adjacent cells specified by the set C_i. In order to quantify the effectiveness of the searcher's movements the probability of nondetection is adopted as a suitable measure. Hence the solution to the problem is a feasible search path of T sequentially adjacent cells which, if followed, minimizes the probability of not detecting the target.

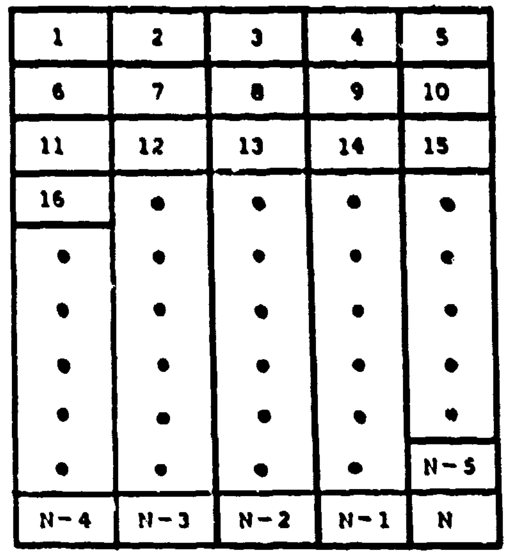


Figure 1.1 Typical Grid and Numbering System.

Describing the problem a bit more graphically, we can imagine the target's initial probability mass dispersed over a collection of cells. The searcher initially located in cell i conducts a thorough search of that cell. If any of the target's probability mass is located within cell i, a percentage of this mass is detected or "cut away". This percentage will vary as a function of the total search effort in the cell and is specified by a detection function known to the searcher. Any target mass outside of cell i is undetected. After the search, all remaining probability mass is relocated on the grid according to the Markov transition matrix. Additionally, the searcher is free to move as long as he remains within the set of adjacent cells C_i. This sequence of searches and movements is repeated for T time periods, with the searcher slowly whittling away at the target's probability mass. At the end of period T, the residual target mass is collected and totalled to yield the overall probability of nondetection.

As will be discussed in the next chapter, the problem may be formulated as a network of N x T nodes, in which nodes represent cells for specific time periods and arcs depict the flow of search effort through the grid. With the objective function defined as the T-time period probability of nondetection and a suitable detection function specified, the problem becomes one of minimizing a convex function subject to network flow constraints. Solutions to this problem involve two cases of interest. The first consists of divisible search effort in which the searcher is allowed to divide the resources available for each time period among several cells. An example of this might be an aircraft engaged in ASW search. Here the aircraft's speed advantage allows him to divide his efforts over a large area. Similarly, a search party consisting of several men might fractionate into smaller groups in order to cover more ground. For this case, the network constraints can be shown to specify a convex feasible region with the resulting problem being solvable by a linearization method. The second case is that of nondivisible search effort and shall be referred to as the integer programming problem. This is more characteristic of searches involving single units such as surface combatants or submarines. Here the searcher cannot divide his resources and speed limitations restrict him to much smaller areas of coverage. This case is much more difficult to address and is solved here with a branch-and-bound algorithm that is presented in Chapter IV.

C. PREVIOUS WORK

This problem has been addressed by several individuals using a variety of approaches. Brown [Ref. 1] proposed a solution for which the search effort was allowed to fractionate infinitely. Within each cell he specified an exponential detection function such that if x units of search effort were placed in cell i where the target is located, the probability of nondetection is given by $\exp(-\beta_i x)$. (Where β_i is the search effectiveness in cell i.) In doing so, he was able to formulate the problem as a convex nonlinear problem and develop an iterative technique for computing optimal search plans. However he allowed no constraints on searcher motion. By constraining searcher movements and using a dynamic programming technique, Eagle [Ref. 2] was able to find an optimal solution to a relatively small integer problem but at the expense of a large amount of computer time. A apparently more efficient heuristic for solving the integer problem was posed by Stewart [Ref. 3] in which a branch-and-bound algorithm employed a modified version of Brown's procedure to calculate a lower bound on trial paths. However Brown's procedure assumes a convex feasible region which is not the case for the integer problem . Thus the "lower bounds" generated are only approximate, and it is possible to incorrectly fathom a branch containing an optimal path. Nonetheless, Stewart believed that near-optimality would be achieved and that this branch-and-bound procedure is probably a good heuristic.

More recently a technique designed by Professors Eagle and Yee [Ref. 4] at the Naval Postgraduate School in Monterey, California, uses the branch-and-bound algorithm as proposed by Stewart yet relies on a better submodel for calculating lower bounds. The submodel used in this technique was developed by Professor Yee. It imposes the constraint on searcher motion but relaxes the constraint on divisiblity of search effort, thereby creating a convex feasible region. This subroutine is very similar to the procedure suggested by Stewart [Ref. 3: pp. 131-2] for solution of the divisible search effort problem and can be shown to yield reliable lower bounds to the integer problem. This paper will explore implementation of this algorithm and propose some possible uses of the procedure.

II. THE OPTIMIZATION PROBLEM

As originally proposed by Stewart [Ref. 3: pp. 130-132], this problem may be set up as network of N by T nodes, where each node represents a particular cell in a specific time period. Borrowing from Stewart's notation, the target's path through the network may be described by the vector $\omega = \{\omega(1), \omega(2), \ldots, \omega(T)\}$, where $\omega(t)$ represents the target's position at time t and p_ω gives the probability that path ω is taken. Search flow through the network is given by x(i,j,t), representing a flow of search effort from cell i in time period t to cell j in time period t+1. The network and some example flows are illustrated in Figure 2.1. Recall that for each time period the searcher is constrained to the cell he previously occupied or any adjacent cells. Let C_j be the set of all cells adjacent to cell j. Then the total search effort in cell j in time t, $\mathbf{X}(j,t)$, is found by summing all flows into the cell as shown in Equation 2.1

$$\mathbf{X}(j,t) = \sum_{i \in C_j} x(i,j,t-1) \quad \text{for } t=2,\ldots,T \qquad \text{(eqn 2.1)}$$

This equation holds for all time periods t except when t=1. The values $\mathbf{X}(j,1)$ are given as an initial conditions for the problem. We will assume that $\mathbf{X}(j,1)=1$ if j is the searcher's starting cell, and $\mathbf{X}(j,1)=0$ otherwise. Using the assumption of an exponential detection function, the probability of target nondetection in cell j during time t is found by:

$$\exp\{-\beta_j \mathbf{X}(j,t)\} = \exp\{-\beta_j \sum_{i \in C_j} x(i,j,t-1)\} \qquad \text{(eqn 2.2)}$$

Here β_j gives the search effectiveness within cell j where $\beta_j \geq 0$. Finally when considering all possible target paths, the probability of target nondetection, Q, after T periods of search is given by:

$$Q = \sum_\omega p_\omega \exp\{-\sum_{t=1}^{T} \beta_{\omega(t)} \mathbf{X}(\omega(t),t)\} \qquad \text{(eqn 2.3)}$$

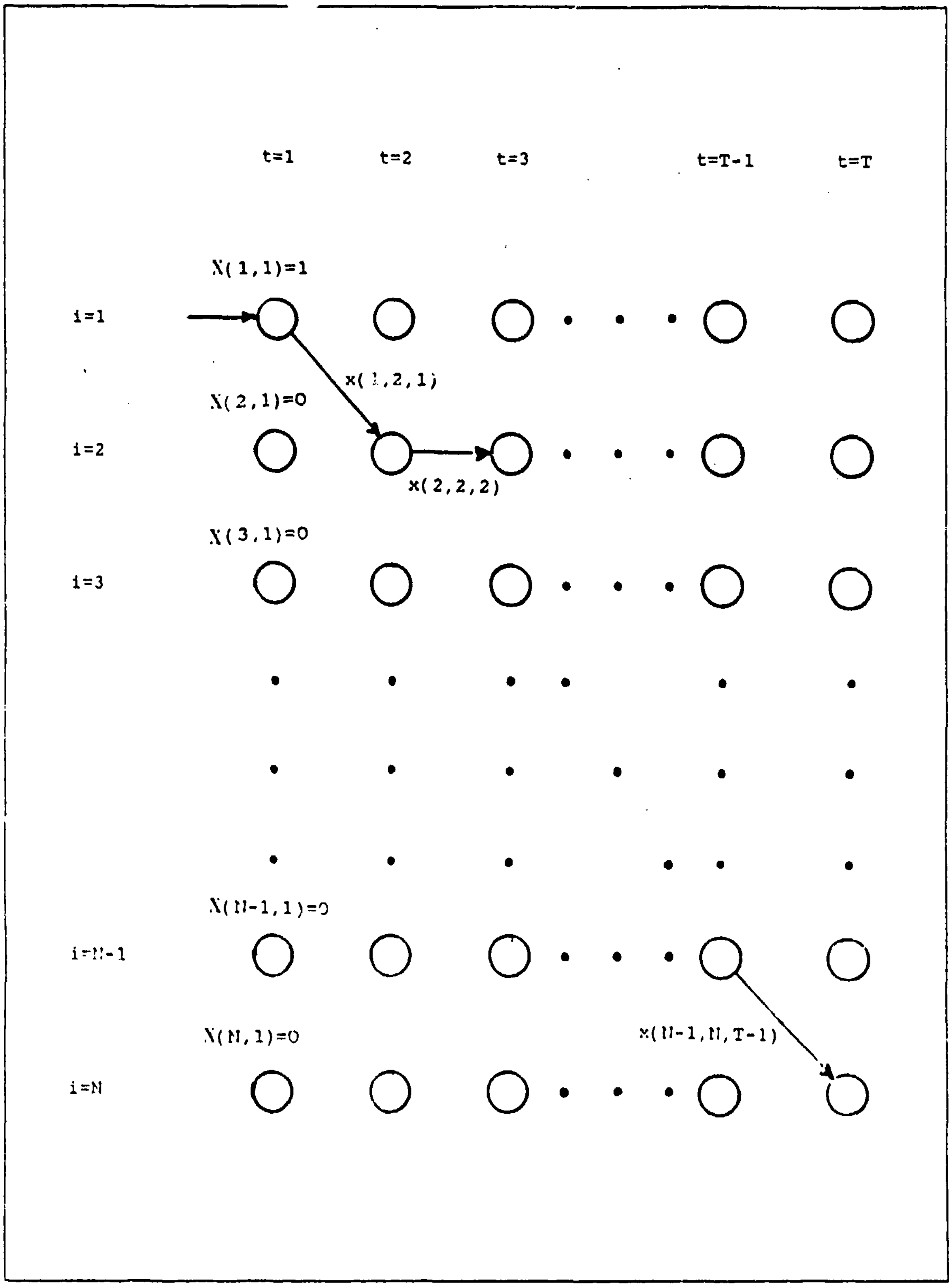


Figure 2.1 Network and Associated Flows.

A possible modification to this is to allow the search effectiveness parameter to be associated with the arc instead of the node. We introduce the parameter α_{ij}, representing the search effectiveness of a flow of effort from cell i to cell j. Initially α_{ij} is assumed to be constant throughout the problem, however the procedure could easily incorporate changes in α_{ij} over time (for instance the parameter could become α_{ijt}). By introducing the above modification and accounting for search in time period 1, the probability Q may be rewritten as:

$$Q = \sum_{\omega} p_{\omega} \exp\{-X(\omega(1),1) - \sum_{t=2}^{T} \sum_{i \in C_{\omega(t)}} \alpha_{i,\omega(t)}\, x(i,\omega(t),t-1)\} \qquad \text{(eqn 2.4)}$$

This is the objective function that a searcher wishes to minimize subject to the constraints of flow balance at each node. Note that there is no search effectiveness parameter included with the search effort for period 1. This is because in time period 1 β_j is assumed to be equal to 1 for all j. The optimization problem is shown below:

Minimize:

$$Q = \sum_{\omega} p_{\omega} \exp\{-X(\omega(1),1) - \sum_{t=2}^{T} \sum_{i \in C_{\omega(t)}} \alpha_{i,\omega(t)}\, x(i,\omega(t),t-1)\} \qquad \text{(eqn 2.4)}$$

subject to :

$$X(i,1) - \sum_{j \in C_i} x(i,j,1) = 0 \qquad i \in \{1, \ldots, N\}$$

$$\sum_{i \in C_j} x(i,j,t-1) - \sum_{k \in C_j} x(j,k,t) = 0 \qquad j \in \{1, \ldots, N\},\ t \in \{2, \ldots, T-1\}$$

$$x(i,j,t) \in \{0,1\}$$

The objective function as written may not be useful for computational purposes because it requires prior knowledge of p_{ω} in addition to the complete enumeration of all possible target paths. For these reasons a computational formula is used within the algorithm which exploits the Markovian nature of the target's motion. Derivation of this formula is shown in Appendix A. It is however useful to present the objective function as shown above because close examination reveals it to be the convex sum of convex terms, or hence, a convex function [Ref. 3: p. 131]. Thus it can be seen that the problem is a nonlinear optimization of a convex objective function. Furthermore, the constraints are linear and highly structured. Specifically, they represent an acyclic network flow problem.

III. THE INFINITELY DIVISIBLE PROBLEM

A. DESCRIPTION OF THE ALGORITHM

Ideally , we would like to solve the search problem for the indivisibility of search effort. In other words, a solution is sought for which x(i,j,t) (and hence **X**(j,t)) are either 0 or 1. However for this case, the feasible region is a set of discrete points in N-space, and the problem is very difficult to solve. If, as Brown suggests the search effort is allowed to fractionate infinitely, the constraints describe a convex feasible region. Then a solution to the infinitely divisible problem may be calculated iteratively by following Stewart's suggestion [Ref. 3: p. 131], of linearizing the objective function and solving the resulting linear program (LP). What makes this procedure feasible is that the LP is an acyclic shortest path problem which can be solved easily and efficiently without the use of a general LP solver.

The method of solution used here for the nonlinear program was first introduced by Frank and Wolfe in 1956 [Ref. 5] and is described below. Given an initial set of feasible flows, the objective function is evaluated. Let this solution point be known as X_1. Next, the objective function is linearized by calculating all possible partial derivatives and then substituting these as edge costs within the network. If Q represents the value of the original objective function and $\hat{Q}$ represents the linearized objective function then, the linear subprogram becomes one of minimizing:

$$\hat{Q}(X) = Q(\mathbf{X}_1) + \nabla Q(\mathbf{X}_1)(X-X_1) \qquad \text{(eqn 3.1)}$$

subject to the network constraints as before (Note: $\nabla Q(\mathbf{X}_1)$ is the gradient of Q evaluated at the point $\mathbf{X}_1$). Because each of the search flow arcs connects time period t with time period t+1, the network can not cycle. Also, since increasing the flow along any arc cannot increase the objective function, all edge cost are nonpositive. So the linear subproblem which solves for $\hat{Q}$ becomes an acyclic shortest path problem with nonpositive costs. Graphically this is shown in Figure 3.1. At the point X_1 the gradient is found and the objective function linearized. In finding the shortest path, the algorithm decreases the value of the linearized objective function until point X_2 at the edge of the feasible region is reached. It is important to note that X_2 is always an extreme point on the feasible region, and that the linearized objective function always underestimates the value of the real objective function. This occurs because a first order Taylor approximation underestimates a convex function. Later this consideration becomes important when a lower bound to the integer solution is sought.

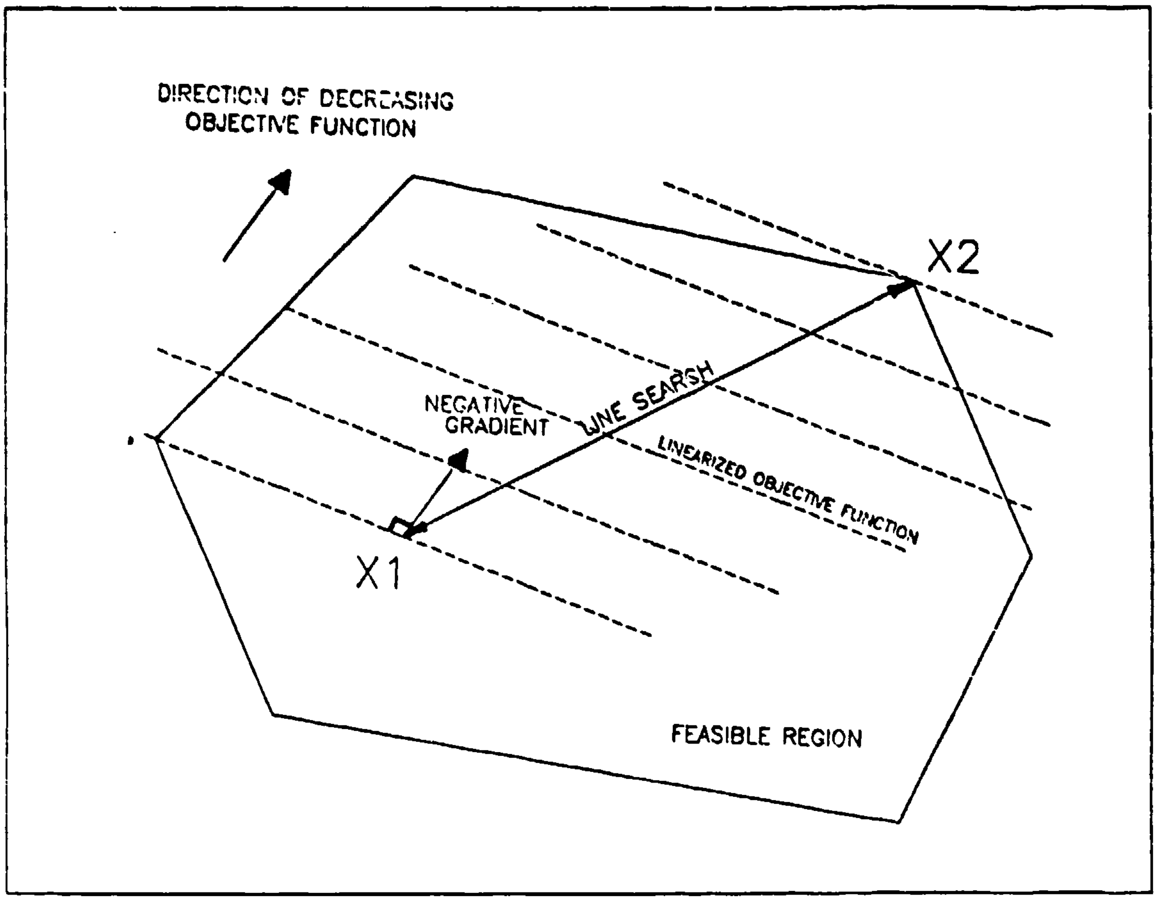


Figure 3.1 Graphical Representation of the Frank-Wolfe Linearization.

The next step is to conduct a line search from X_1 to X_2 for the point that minimizes the original objective function Q. After updating X_1 with the flows defined by the minimizing point from the line search, the procedure repeats itself until some stopping criteria is met. This iterative technique for solving the infinitely divisible problem is essentially the same as proposed by Stewart with the exception that there are no upper bounds placed on flow efforts x(i,j,t). The importance of this relaxation is that it maintains the linear subproblem as an acyclic shortest path problem, which is perhaps the easiest of all nontrivial LP's to solve.

B. IMPLEMENTATION OF THE DIVISIBLE SEARCH EFFORT PROCEDURE

The divisible search effort algorithm was coded in Fortran and run on the Naval Postgraduate School's IBM 3033 mainframe computer. The program was written in general fashion such that minimal changes are required to run problems of various sizes. Because the infinitely divisible program is a subprogram of the branch-and-bound solution, much time and effort was spent to develop an efficient algorithm. For this reason the program makes extensive use of subroutines and special data structures such as adjacency lists [Ref. 6: pp. 200-1] in hopes of obtaining efficiency with minimal storage requirements. Specific details of the program are provided along with a program listing in the Appendix B. This section will serve merely as a synopsis of the salient features of each implementation.

An initial feasible solution is input to give a starting point for the algorithm. This point, $\mathbf{X}_1$, consists of a set of flows associated with the searcher remaining in his starting cell for the entire T time periods. Given this set of flows, the probability of nondetection, labelled PND_1, is calculated via the computational formula listed in Appendix A. Additionally the probability of nondetection can be divided into "reach" and "survive" probabilities (also presented in Appendix A) which are used to calculate the partial derivatives at $\mathbf{X}_1$. Once these partial derivatives are found and the objective function linearized, a simple dynamic programming technique is used to find the shortest path through the network to yield the extreme point $\mathbf{X}_2$ that minimizes the linear objective function. A quadratic line search is then used to find the minimum probability of nondetection along the line from the start point $\mathbf{X}_1$ to the extreme point $\mathbf{X}_2$. This new point becomes $\mathbf{X}_1$ for the next iteration and the whole procedure repeats itself until the stopping criteria is met. This procedure is illustrated in the flowchart of Figure 3.2.

Frank and Wolfe showed that a lower bound on the optimal objective function value can be obtained at each iteration. This is important since true lower bounds are required for use in the branch-and-bound procedure. From Figure 3.3 we see that:

$$PND(\mathbf{X}^*) \geq PND(\mathbf{X}_1) + \nabla PND(\mathbf{X}_1)\ (\mathbf{X}^* - \mathbf{X}_1) \qquad \text{(eqn 3.2)}$$

By the convexity of $PND(\mathbf{X})$. And furthermore:

$$PND(\mathbf{X}^*) \geq PND(\mathbf{X}_1) + \nabla PND(\mathbf{X}_1)\ (\mathbf{X}_2 - \mathbf{X}_1) \qquad \text{(eqn 3.3)}$$

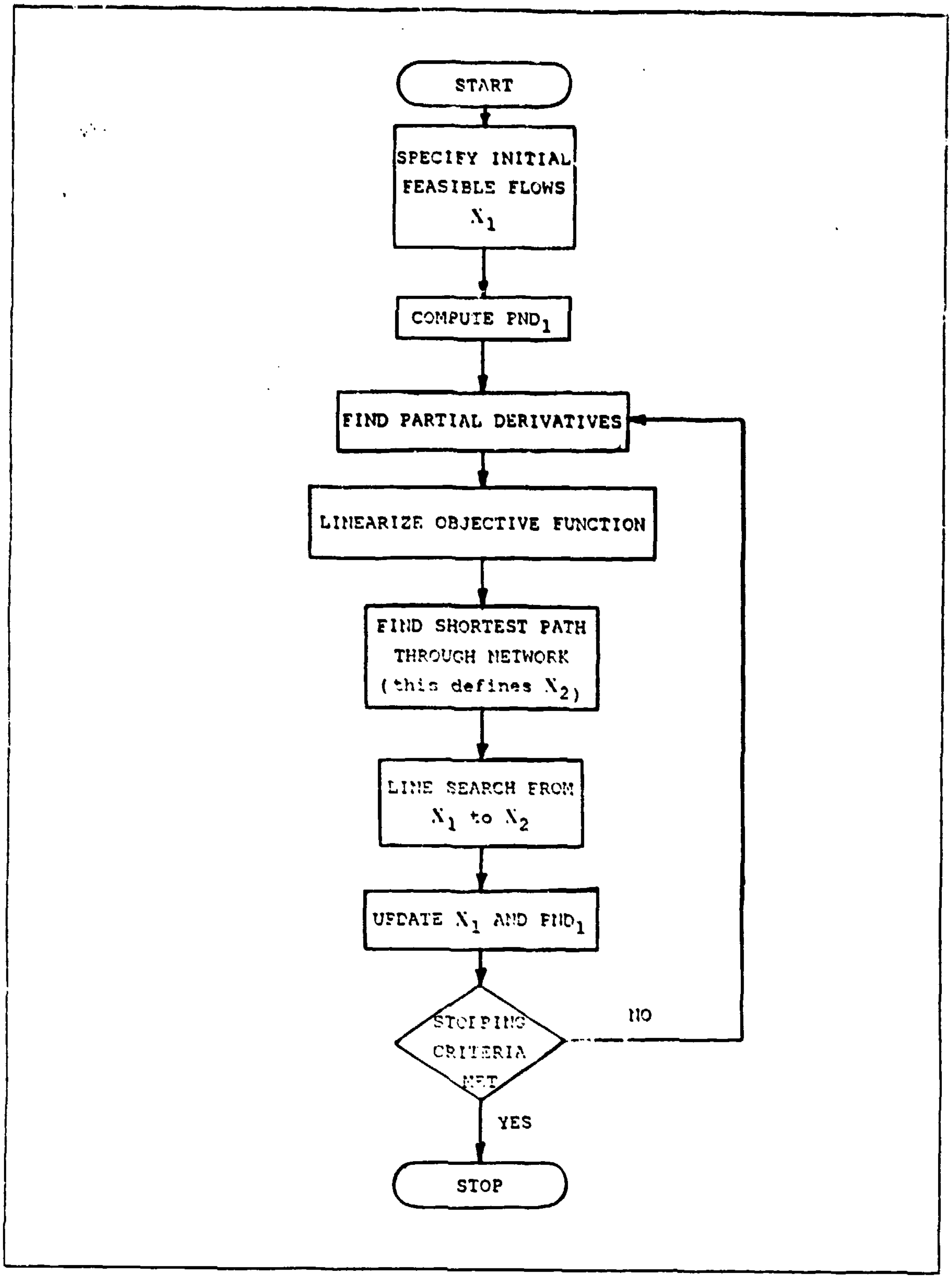


Figure 3.2 Flowchart for Infinitely Divisible Search Effort Problem.

since X_2 minimizes the linear subproblem objective $\nabla PND(X_1)\ X$ subject to the required network constraints. So

$$DELTA = -\nabla PND(X_1)\ (X_2 - X_1) \qquad \text{(eqn 3.4)}$$

shown in Figure 3.3 is an upper bound on how much improvement is possible if the nonlinear procedure is continued. The procedure was stopped when DELTA became sufficiently small.

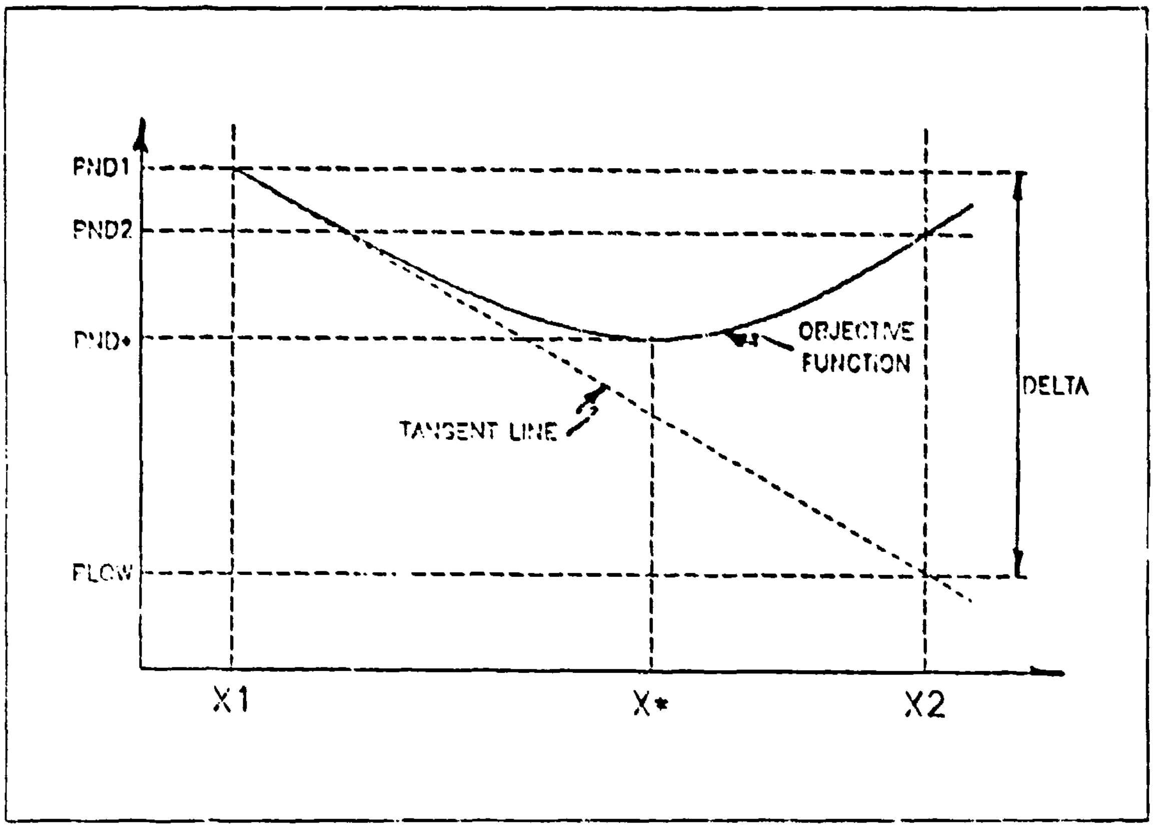


Figure 3.3 The Lower Bound Shown Graphically.

The resulting program was run successfully on several example problems of various sizes including a 15 x 15 cell grid with 25 time periods. Specifics of this problem will be discussed in Chapter V. All cases that were considered involved situations in which the target's mass was initially located at a point and then allowed to spread uniformly in all directions. This type of search is commonly referred to as a datum search with the target's starting point known as "datum".

Throughout the testing and evaluation of this program several key items were observed:

- The Frank-Wolfe method resulted in fast initial convergence as evidenced by a large drop on the probability of nondetection after just one iteration. When close to the optimal solution, convergence was much slower.
- For each Frank-Wolfe iteration, the start point probabilities (PND_1) and the lower bound probabilities at the extreme point (PLOW) followed a definite pattern as shown in Figure 3.4. This observation becomes important later when considering early termination of the lower bound calculation in the branch-and-bound algorithm.
- As optimality was approached the minimum value of the objective function obtained from the quadratic line search between $\mathbf{X}_1$ and $\mathbf{X}_2$ moved closer to $\mathbf{X}_1$. This seems somewhat intuitive when considering that the algorithm is continuously stepping towards the optimal point.
- The algorithm was relatively quick; an important consideration for the branch-and-bound problem, and suggested that larger problems could be solved for the case of divisible search effort.
- For the datum searches it was interesting to note that as the problem started the search effort did not fractionate but instead moved off directly towards the target's datum. As the problem proceeded, the search effort began to divide and disperse once the searcher was located on top of the target.

C. SUMMARY

The divisible search effort problem was found to be solvable by the Frank-Wolfe method which consists of the following steps: linearizing the objective function; solving the network shortest path problem to find an extreme point; and then conducting a line search from start point to extreme point. Each time an extreme point is discovered, a lower bound to the solution is available through use of a Taylor first order approximation. The divisible search algorithm was found to run quickly for relatively large problems (run times for various problems will be given later). This was extremely promising, for if a legitimate real world scenario can be modelled using this method, practical implementation of this program may prove to be fruitful. Although specific applications are not covered in this study, several possible scenarios are presented in Chapter VI.

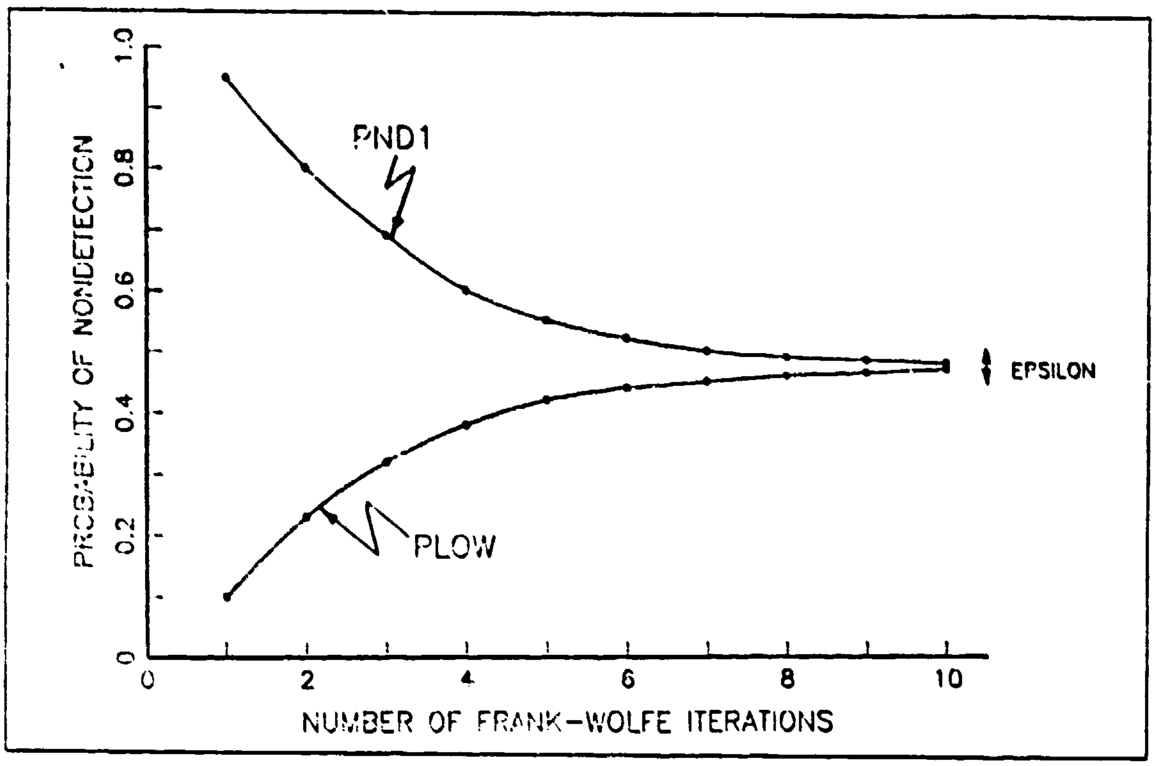


Figure 3.4 Convergence of Probabilities During Frank-Wolfe Iterations.

IV. THE INTEGER PROGRAMMING PROBLEM

A. DESCRIPTION OF THE ALGORITHM

As previously discussed, it is the integer problem for which a solution is desired. Yet this problem is difficult to solve and grows in complexity very quickly as the number of cells and time periods increase. If we were to consider the set of all possible search paths these might be displayed as a tree like the one shown in Figure 4.1 for a nine cell problem. For a searcher starting in cell 1, for time period 2 he may proceed to any cell in $C_1 = \{1, 2, 4\}$. Rather than enumerating all possible paths through C_1, we would like to consider each path individually as a trial path and then systematically discard or "prune" trial paths that are unacceptable. This may be accomplished by a branch-and-bound algorithm like the one described by Stewart [Ref. 3: pp. 133].

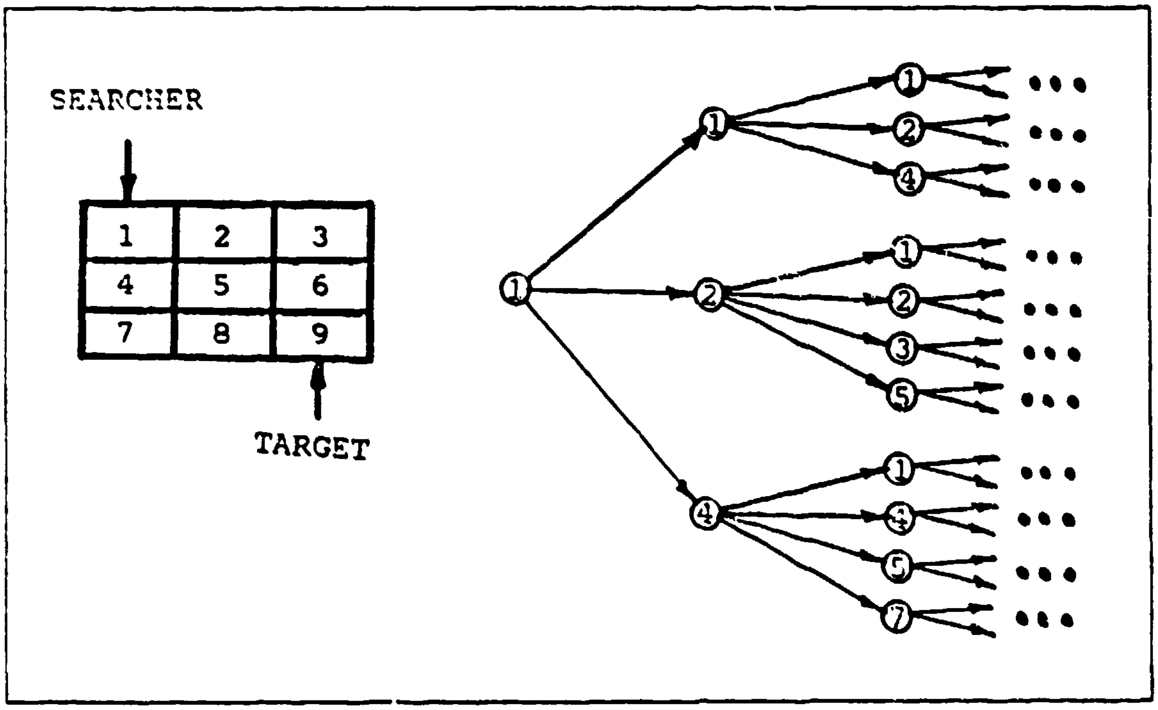


Figure 4.1 Tree Representing Possible Search Paths for a Nine Cell Problem.

A branch-and-bound algorithm compares a lower bound for a given trial path with the current best (ie., smallest) probability of nondetection, called PBEST.

(Initially PBEST is obtained from a user provided feasible solution.) If the lower bound is greater than PBEST, the trial solution is "fathomed". This occurs because the best solution attainable with the proposed trial path is always worse than the current solution. On the other hand, if the computed lower bound is less than PBEST, the trial path can not be fathomed, for there may exist a subset of that trial path that will yield a probability of nondetection smaller than the current best. In this case the trial path must be further specified by stepping deeper into the tree, computing a new lower bound, and then continuing the same procedure as discussed above.

These points are best illustrated with an example. Consider the nine cell problem discussed above. Let the first trial path be specified as {1, 2}. This represents a the set of all possible integer paths starting in cell 1 in time period 1 and proceeding to cell 2 in time period 2 (shown by the middle branch of the tree in Figure 4.1). The lower bound PLOW for this trial path is calculated and compared with the current PBEST. For PLOW greater than PBEST, there exist no paths of the sequence {1, 2, . . .} that will yield a solution better than PBEST. Therefore the trial path would be fathomed. If PLOW is found to be less than PBEST there may exist a path of the form {1, 2, . . .} with a probability of nondetection less than the current PBEST. We cannot fathom this path but instead must step deeper into the tree to examine the set of all trial paths specified by the set $\{1, 2, j \in C_2, \ldots\}$. For each of these paths, lower bounds will be calculated and compared to PBEST, resulting in fathoming or further branching. Whenever branching results in the complete specification of an integer solution, the probability of nondetection is calculated, compared with PBEST, and the current solution updated as necessary. After all trial paths of the form $\{1, 2, j \in C_2, \ldots\}$ are "considered" (ie., fathomed or completely enumerated), the algorithm steps "out" to consider other trial paths of the form $\{1, j \in C_1, \ldots\}$. When all possible paths through C_1 are considered the algorithm is finished. With this in mind the only remaining complication is the calculation of the lower bound for various trial solutions.

Suppose for a T-period problem a trial path is specified for first t time periods. This leaves $T-t$ time periods of search over which the probability of target nondetection may be minimized. Slightly modifying Stewart's notation [Ref. 3: p.134], this probability may be written as the product of two terms:

$$\text{Prob}\{\text{nondetection by time } t\}$$

and

$$\text{Prob}\{\text{nondetection in periods } t \text{ to } T \mid \text{nondetection by time } t\}$$

Given the integer solution thru time t, the Prob{nondetection by time t} is a constant. Therefore in order to minimize the overall probability of nondetection, the second term must be minimized. Or in the case of the branch-and-bound problem, a lower bound may be obtained from this term. By allowing the search effort to fractionate from time $t+1$ until time T, the problem becomes an infinitely divisible problem of $T-t$ time periods. As previously discussed, a lower bound may be obtained via the first order Taylor approximation that results from the Frank-Wolfe method. Hence a lower bound on the integer trial path is available.

Summarizing the branch-and-bound steps as discussed above: A trial path is generated which specifies an integer solution for the first t time periods. Based on this trial path we must update the target's probability distribution, accounting for those first t periods of search and target transitions. Next a subroutine is called where the search effort is allowed to fractionate for the remaining $T-t$ periods in order to find a lower bound, PLOW, for the trial path. Comparing this PLOW to the current PBEST, the trial path is either fathomed or further branching is undertaken. This continues until all possible trial paths are fathomed or completely enumerated. A flowchart showing the basic integer algorithm is shown in Figure 4.2.

B. IMPLEMENTATION OF THE INTEGER PROGRAMMING PROCEDURE

Once the divisible search effort problem was available, the branch-and-bound procedure could be implemented fairly easily. Like the previous program, this procedure makes extensive use of subroutines and adjacency lists. A set of nested "do loops" is used to control the generation of trial solutions and associated branching. For each trial path, a modified divisible search effort program is called to find the lower bound. As previously stated, for every call to the subroutine the time horizon and the target probability distribution must be updated. In addition, an initial set of feasible flows must be generated to span the reduced time horizon within the subprogram. This initial solution is achieved as before by letting the searcher remain in the same cell for all $T-t$ time periods. The subprogram returns a value of PLOW which is used to determine whether fathoming or further branching is appropriate. This procedure continues until all possible paths are "considered".

Initially the branch-and-bound algorithm as described above was tested on a small 4 cell, 3 time period problem where all calculations were verified by hand. The next implementation was a 9 cell problem with 10 time periods like the one used by Eagle [Ref. 2: pp.1113-4] in which the searcher starts in cell 1 and the target begins in

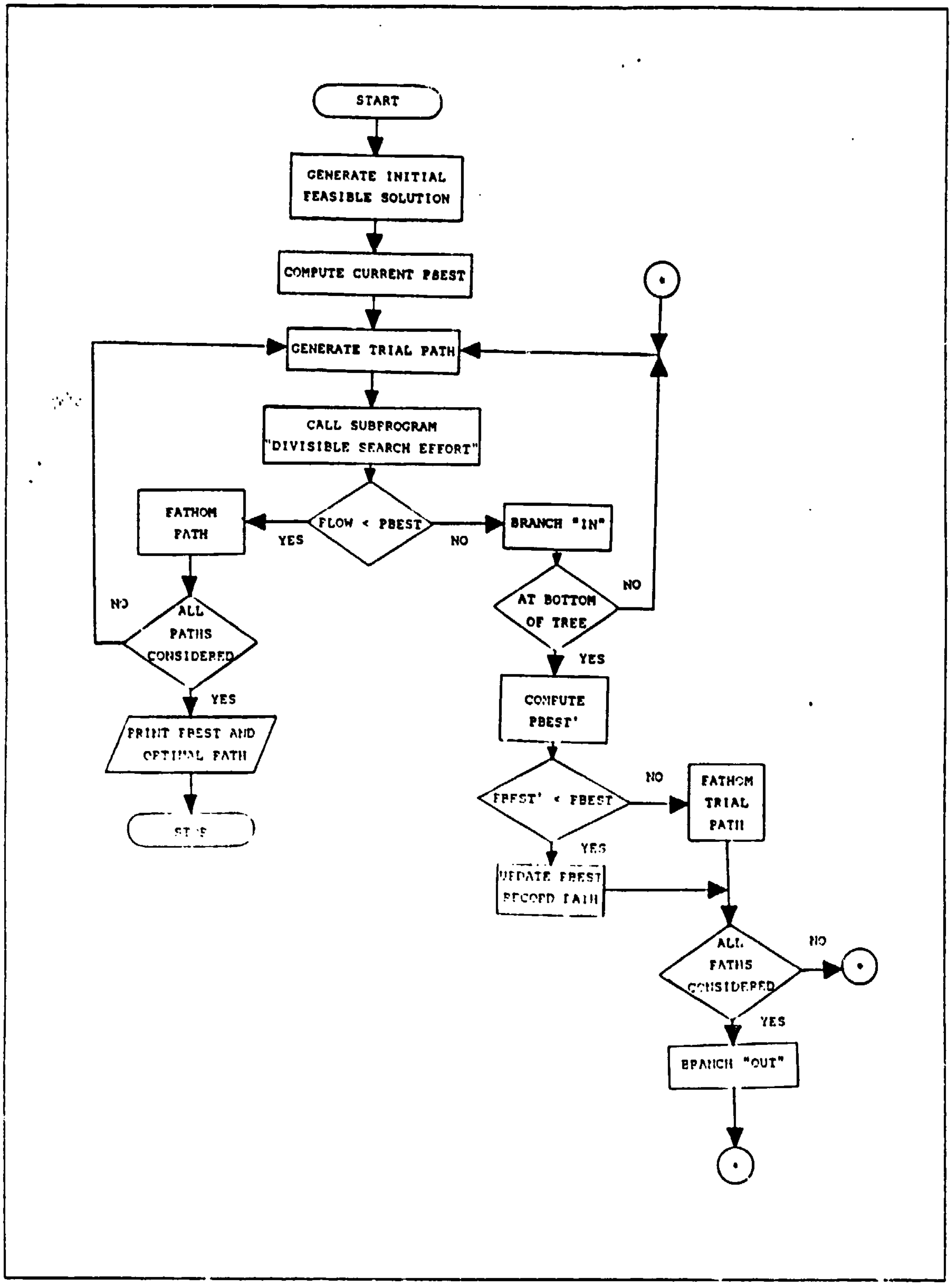


Figure 4.2 Flow Chart Showing Integer Solution Procedure.

cell 9. In this problem the Markov transition probabilities are given as follows: the target remains in the cell he currently occupies with probability .4, while the remaining probability is divided equally among all adjacent cells. For this case adjacent cells are those that share a common side, thus diagonal movements by searcher and target are not allowed. Eagle was able to compute optimal search paths for this problem using a dynamic programming technique, yet at the cost of 19 minutes of computer processing time.

The first attempt at this problem using the branch-and-bound technique was conducted with a starting solution of cell 1 for all 10 time periods. Additionally, for each call the subprogram was allowed to run until the lower bound was known to within a user defined interval. This approach took far too much computer time. In fact, an optimal solution was not obtained after 15 minutes of run time. Several improvements were necessary in order to cut down this time requirement to a reasonable one. These are listed below:

- Using Eagle's optimal paths, branching discipline was improved such that trial paths closest to the optimal path were considered first. In this way better lower bounds were achieved quicker resulting in more efficient fathoming and therefore fewer trial paths to consider. Although this required knowledge of the actual optimal paths, improvements are still available by implementing at least some sort of branching discipline possibly arrived at through a best guess of the optimal path.
- Starting solutions were improved by using a best guess of the optimal path. With a near-optimal starting solution, a better PBEST is available. This also results in better fathoming of nonoptimal trial paths.
- The subprogram was stopped before finding the lower bound within a small window. This was accomplished via two important changes with the result that overall less time was spent in searching for lower bounds. Recall the trend of converging probabilities in the subprogram as illustrated in Figure 3.4. This same example is shown again in Figure 4.3 with a few additions. Notice how bounds on the probability of nondetection associated with a given trial path converge until the difference is less than a user defined value, ε. Rather than allowing this to occur, the subroutine may be stopped as soon as PLOW is greater then the current PBEST, or as soon as PND_1 is less than PBEST. First, suppose the current PBEST is given by $PBEST_1$ in Figure 4.3. As soon as PLOW exceeds PBEST we know that for the trial path of consideration, the best possible solution will always be worse than the current solution. Therefore the trial path can be immediately fathomed. Suppose on the other hand that the current PBEST is given by $PBEST_2$. It can be seen that as soon as PND_1 is less than $PBEST_2$ that this path cannot be fathomed (PLOW for this path will never exceed $PBEST_2$). For either case, continued iteration towards a better

lower bound is unnecessary. The computations may be halted and branching or fathoming should occur. These last two improvements were *significant* in reducing the number of Frank-Wolfe iterations and hence the time requirements for the integer algorithm.

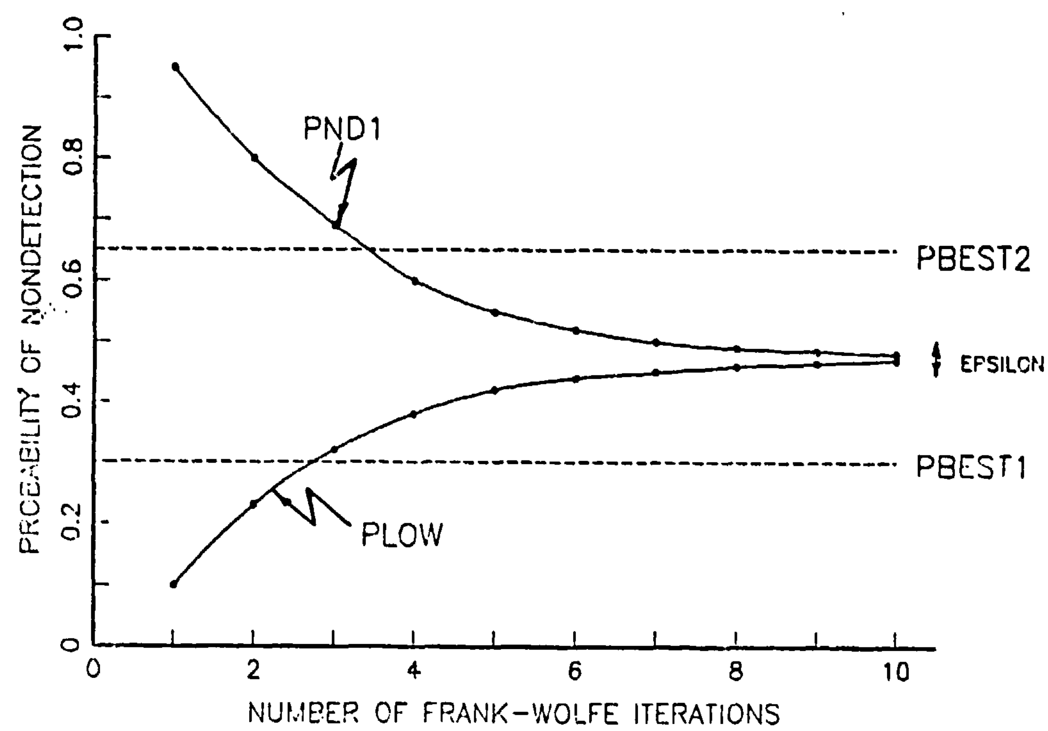


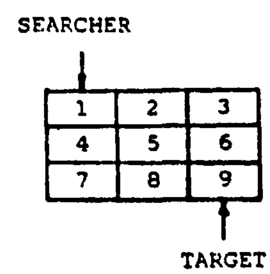
Figure 4.3 Stopping the Lower Bound Calculation Early.

Outside of these improvements a few others were made to try and cut off more time. Recall that each time the objective function is linearized and the shortest path found, the resulting extreme point specifies an integer solution. If the probability of target nondetection associated with this point is better (ie., smaller) than the current PBEST, this solution can be stored and PBEST updated. This provides some reduction in the time required to get a near-optimal solution, resulting in a lower PBEST and therefore more efficient fathoming. Additionally, this improvement may be used to help generate initial feasible solutions. At the begining of the algorithm the subprogram may be called with an uncorrected time horizon and allowed to run several Frank-

Wolfe iterations. For each iteration the extreme point solution is checked and the best one recorded. In this fashion a good starting solution is easily obtained. Even though this procedure adds some extra time to the algorithm, this time is well spent, especially for problems where a near-optimal starting path is not easy to estimate.

C. SUMMARY

After implementation of the improvements and modifications the program was allowed to run on the nine cell problem with the searcher starting in cell 1 and the target starting in cell 9. Multiple optimal search paths with probabilities of nondetection equal to .4219 were found after 112 seconds of computer run time. These paths are listed in Figure 4.4.



PROBABILITY OF NONDETECTION = .6444

1	2	5	8	9	6	5	8	5	5
1	2	5	6	9	8	5	6	5	5
1	4	5	8	9	6	5	8	5	5
1	4	5	6	9	8	5	6	5	5

Figure 4.4 Optimal Integer Paths for a Nine Cell Problem of Ten Time Periods.

These answers are somewhat different from those found by Eagle's dynamic programming approach because of basic differences between the structures of the two models. Eagle's dynamic programming model allowed for a target transition before the first search took place; the branch-and-bound algorithm accounted for the first search and then allowed a target transition. This resulted in an extra period of search for the dynamic programming method causing slightly different search paths to be found by the two procedures. Also, the dynamic programing approach did not use an exponential detection function within each cell but instead set the probability of detection equal to one if the target and searcher occupied the same cell simultaneously. Hence, the probabilities obtained by the dynamic programming solution are lower than those presented by this paper. Despite these differences the key item of significance is the great reduction in computer run time for the branch and bound algorithm as opposed to the dynamic programming technique. For this problem it was an order of magnitude decrease. It is also interesting to note that for this seemingly small problem there are some 400,000 possible searcher paths, of which only 3,940 were actually considered as trial paths by the branch-and-bound algorithm.

V. APPLICATIONS

A. INTRODUCTION

Following successful implementation of the two algorithms, several applications were run on problems of various sizes. Some of these cases are listed in Table 1 along with amplifying data regarding computer run times and for integer solutions, the number of trial paths considered. To provide an example of the size and scope of solvable problems, two instances from Table 1 will be discussed in detail. For the divisible search effort case, a 15 by 15 problem with 25 search periods is examined, while a smaller 7 by 7 grid with 10 search periods is used to present the integer application. Note that all problems discussed here are applications involving datum searches of similar geometry and that all search effectiveness parameters (α_{ij}) were set equal to 1 for simplicity. This is important when considering the test results as shown below, for the use of other geometries and encounters will undoubtablely result in significantly different run times. This will be discussed in more detail in the last section of this chapter.

B. A DIVISIBLE SEARCH EFFORT APPLICATION

As stated above, a datum search on a 15 by 15 grid of cells with 25 time periods was solved with the divisible search effort algorithm. This problem involves 44,376 arcs and several hundred thousand possible searcher paths. Despite this, the algorithm ran very efficiently and gave no indication of being anywhere near the upper limit on solvable problem size.

Again this application like all others presented so far, involved a datum search. But this time, instead of the target starting in a corner cell, he was initially placed at the very center of the grid while the searcher started in cell 1 in the upper left hand corner. For the target, the Markov transition matrix was chosen to allow him to disperse uniformly in all directions. Within any given cell he stayed with a probability of .4, with the remaining probability being distributed evenly among cells sharing a common side. Diagonal movements by the searcher were allowed. The solutions are illustrated best by Figures 5.1 through 5.8. Notice as the problem starts the searcher keeps all of his resources together as a single unit and begins to march off towards the target's starting cell. Later in time period 6, the searcher is two diagonal squares away

TABLE 1

RUN TIMES FOR VARIOUS PROBLEMS

GRID SIZE	TIME PERIODS	NUMBER OF ARCS	DIVISIBLE PROBLEM RUN TIME (SEC)	INTEGER PROBLEM RUN TIME (MIN)	INTEGER PROBLEM NUMBER OF TRIAL PATHS
3x3	10	297	4.7	1:52	3940
5x5	10	1521	4.3	3:40	1469
7x7	10	3249	12.0	4:40	945
15x15	25	44376	295.0	–	–

from datum. For the next period the optimal allocation of his search effort is for him to divide his resources. By time period 8, he is on top of datum with his search effort divided among three cells; but this time the division is more evenly distributed with the largest portion centered over datum. In the next period, the searcher fractionates his efforts even more, but curiously there is a smaller portion in the center cell and a greater concentration in the surrounding cells. It almost looks as though the searcher is trying to catch up to the ever-expanding probability mass of the target. The one exception to this is the set of cells along the searcher's previous track; probably because of the lack of undetected target mass in these cells. For the next block of time periods, the searcher's efforts become dispersed somewhat symmetrically as shown for period 16 in Figure 5.7, but this time the largest fraction of effort remains centered on datum. This is not totally surprising when considering that this cell will always contain the biggest part of undetected mass because of the target's starting position and the Markov transition matrix as defined. We might picture the searcher perched atop the target's mass distribution, slowly carving away at the small peak at the center of the grid. Also note that in time period 16 the distribution of search effort is not wholly symmetrical; the cells in the upper left section contain much smaller amounts. Again this is because as the searcher initially came onto datum he thoroughly sanitized his track leaving very little target mass in these cells. Now, as time proceeds the target's undetected mass will slowly filter back over the track. Yet this amount of mass is so small relative to other cells that the optimal allocation of search effort does not include much coverage of this area. The allocations of effort change more slowly as the problem continues. As before, search effort within the area of coverage remains concentrated in the center and more sparsely distributed near the edges. However of interest is the fact that the total area of coverage does not change from time period 16 to time period 25. The searcher has essentially moved to the center of datum, dispersed his effort and remained in the same spot for the duration of the problem. In doing so he achieved an overall probability of nondetection equal to .6142 which may seem surprisingly high. But recall that the target was afforded eight time periods of "escape" before the searcher reached datum.

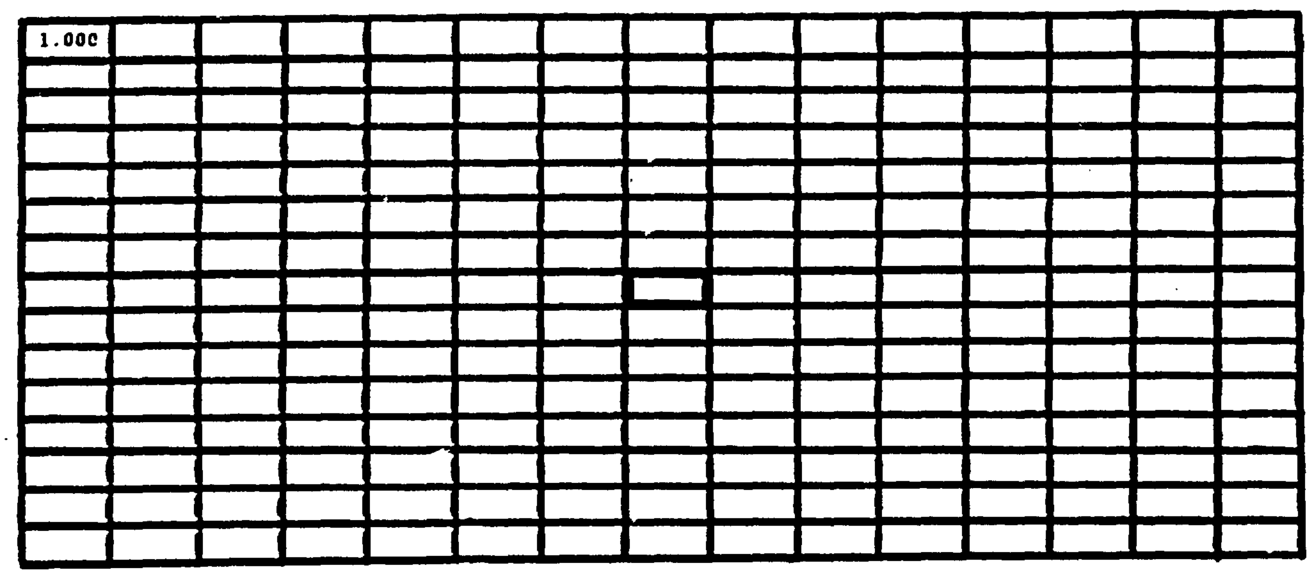


Figure 5.1 Allocation of Search Effort for Time Period 1 on a 15 by 15 Grid.

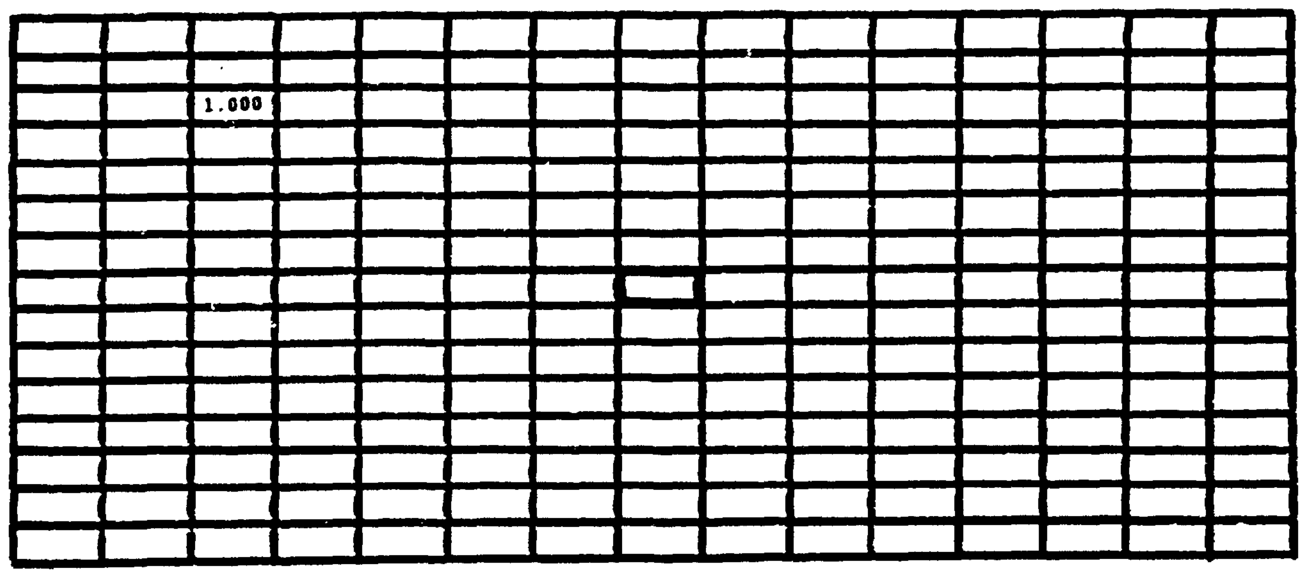


Figure 5.2 Allocation of Search Effort for Time Period 3 on a 15 by 15 Grid.

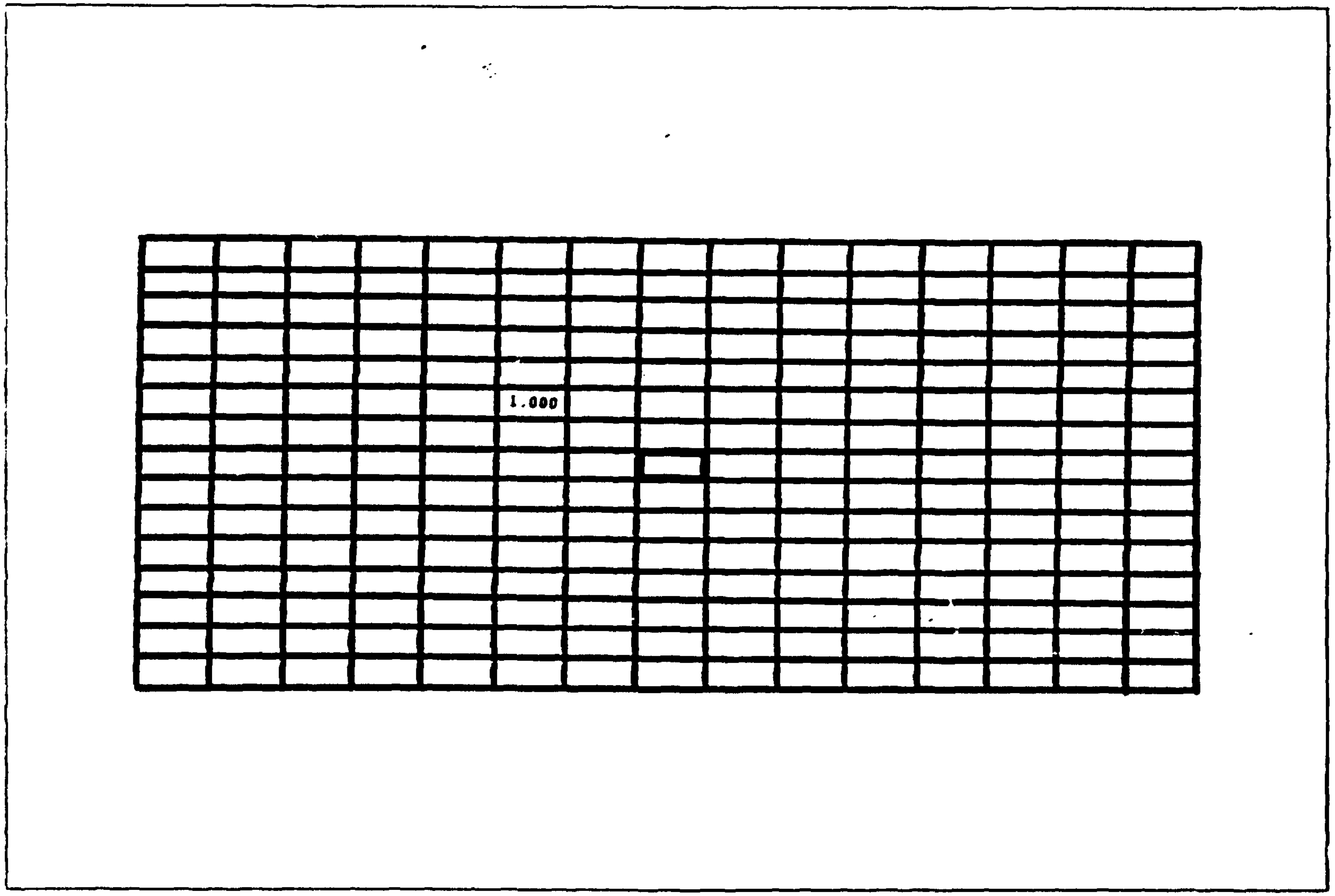


Figure 5.3 Allocation of Search Effort for Time Period 6 on a 15 by 15 Grid.

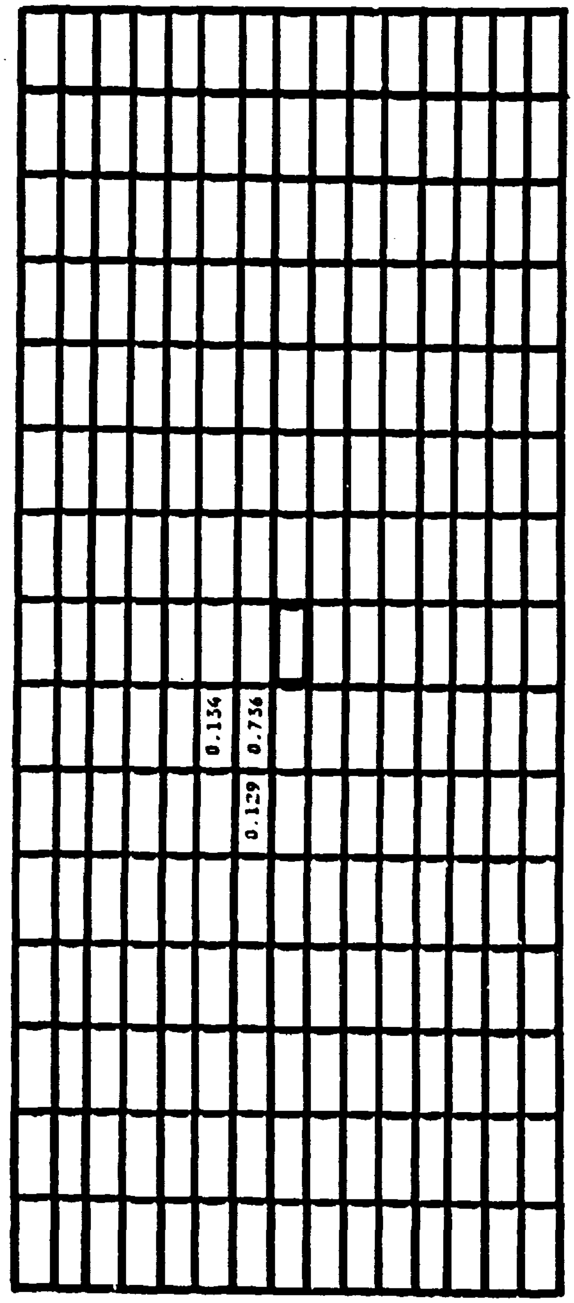


Figure 5.4 Allocation of Search Effort for Time Period 7 on a 15 by 15 Grid.

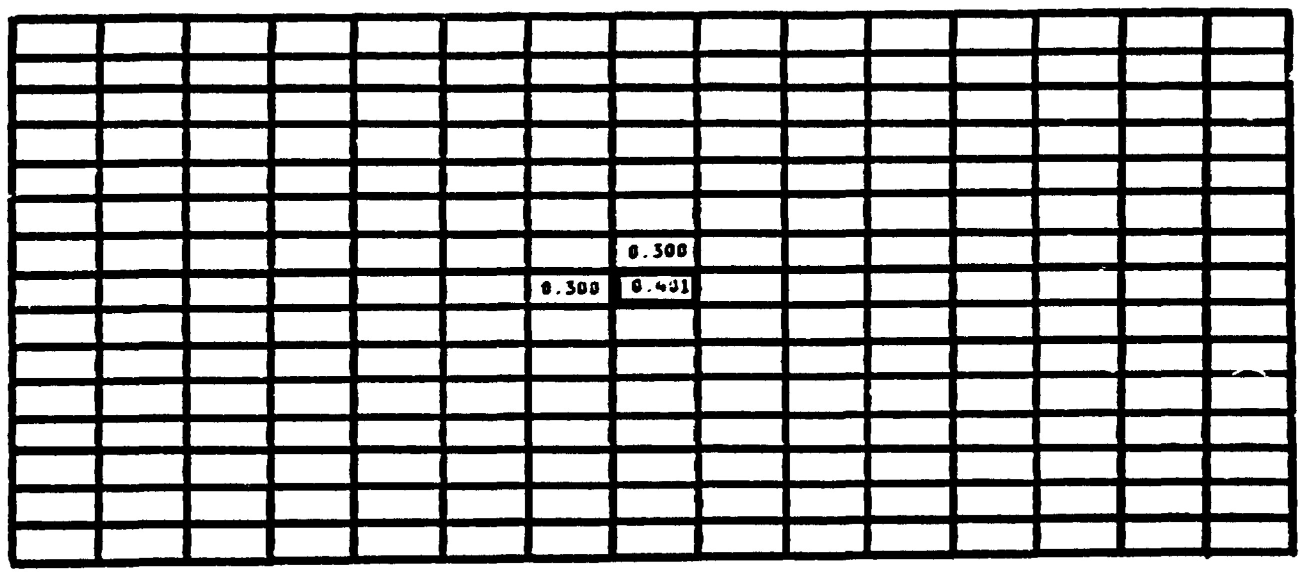


Figure 5.5 Allocation of Search Effort for Time Period 8 on a 15 by 15 Grid.

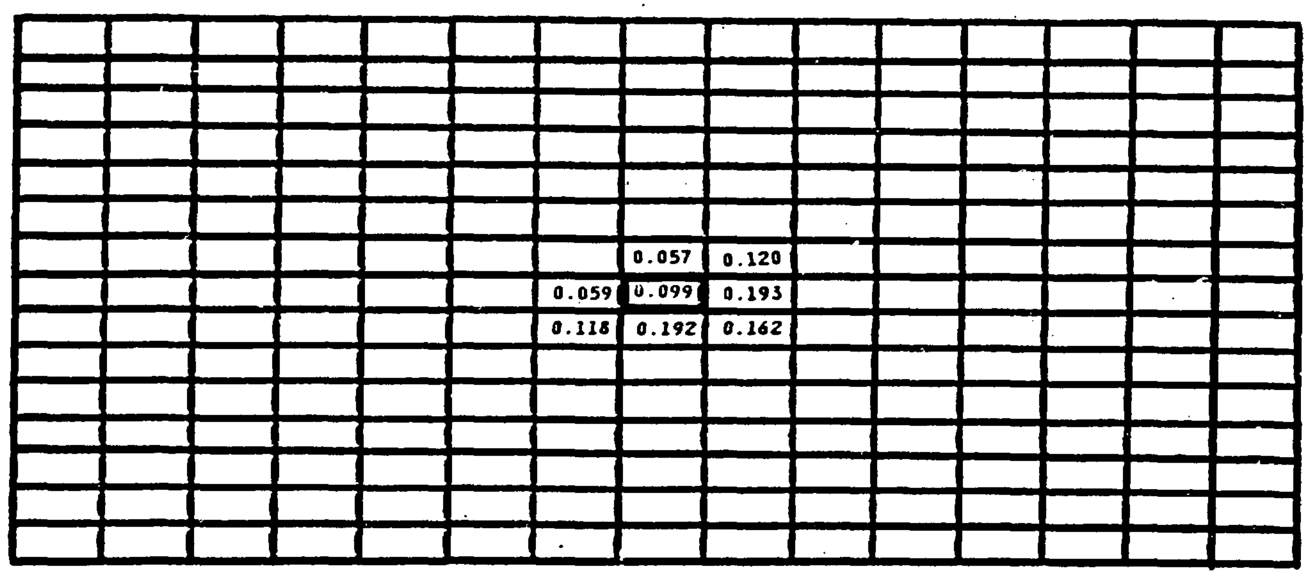


Figure 5.6 Allocation of Search Effort for Time Period 9 on a 15 by 15 Grid.

						0.022	0.085	0.026						
					0.006	0.053	0.055	0.067	0.019					
					0.053	0.045	0.089	0.057	0.064					
					0.028	0.072	0.051	0.069	0.028					
						0.027	0.055	0.030						

Figure 5.7 Allocation of Search Effort for Time Period 16 on a 15 by 15 Grid.

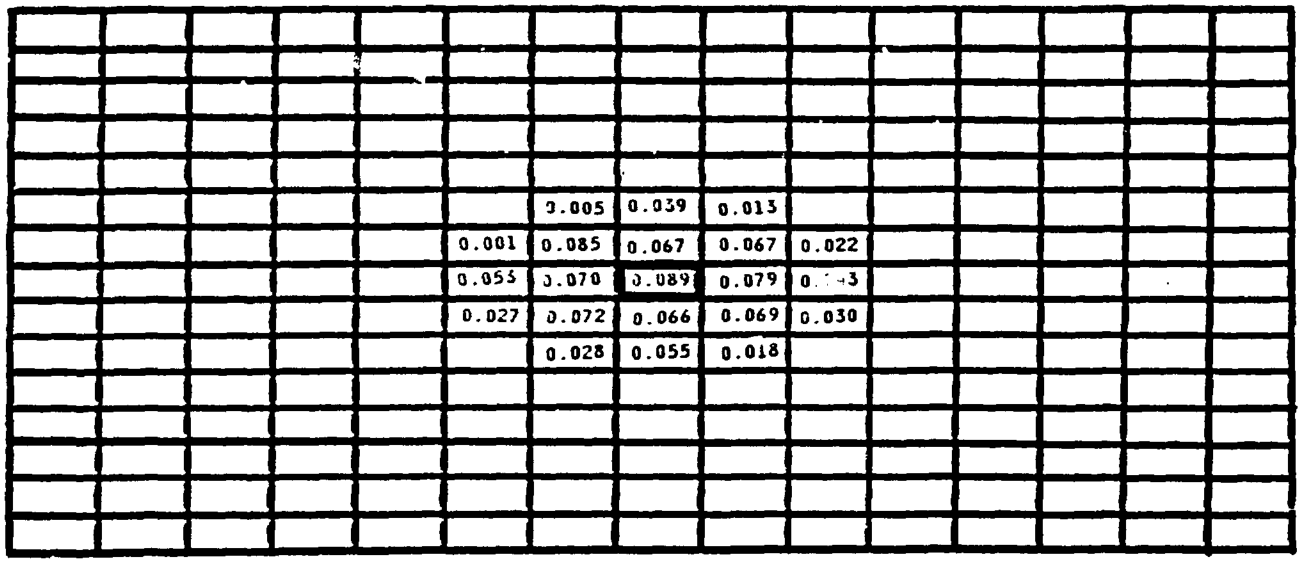


Figure 5.8 Allocation of Search Effort for Time Period 25 on a 15 by 15 Grid.

C. THE INTEGER APPLICATION

A smaller problem is used to demonstrate the integer application. For this case a datum search of a 7 by 7 grid for 10 time periods is discussed. The target, initially in the lower right hand corner of the grid, transitions with the exact same probabilities as presented in the divisible search effort problem of Section B. A searcher starting in cell 1 is once again permitted to move diagonally within the grid. Figure 5.9 shows part of the solution output.

As the procedure begins, a user-input feasible path is used to calculate the initial probability of nondetection as shown in the first line of Figure 5.9. This starting solution is improved upon by allowing several Frank-Wolfe iterations to occur in the divisible search effort subprogram. For each extreme point solution generated the probability of nondetection is calculated and the best one recorded. This best solution becomes the updated PBEST shown in line 2.

With this new starting solution the branch-and bound procedure begins. The first trial path is {1, 1}. After five Frank-Wolfe iterations (listed under the column heading "FW" in Figure 5.9), the path is fathomed because the calculated lower bound PLOW was greater than PBEST. Fathoming of this path is significant because literally thousands of trial paths of the form {1, 1, C_1, . . .} are immediately pruned from the tree. Next the path {1, 2} is considered. In this case 4 Frank-Wolfe iterations occurred until the start point probability PND_1 was found to be less than PBEST. Based on this we known that the lower bound for trial path {1, 2} will never be greater than PBEST and therefore the path will never be fathomed. Further iterations are unnecessary and branching must occur. Now each path of the form {1, 2, C_2} is considered. For the first five cases fathoming occurs until trial path {1, 2, 10} where PND_1 is greater than PBEST. We may not fathom, but must branch again. The program continues fathoming and branching until all possible trial paths are considered. This resulted in the generation of 945 trial paths before the procedure was completed ending with the discovery of two optimal integer paths shown in Figure 5.10. The probability of nondetection for both paths was .6444. Note that the two paths are symmetrical to each other and that, like the divisible search effort application, the optimal path has the searcher initially speeding off towards datum and then conducting a search about that area. Again for this case, the probability of nondetection seems somewhat high, but recall that the target has been given ample opportunity to disperse.

INITIAL PBEST= .7245 1 9 17 25 33 41 49 49 49 49
UPDATED PBEST= .6573 1 9 17 25 33 41 42 48 40 41

PBEST	PND1	PLOW	FW	TRIAL PATH
.6573	.6655	.6593	10	1 1
.6573	.6550	.4656	3	1 2
.6573	.7118	.6747	4	1 2 2
.6573	.7400	.7020	4	1 2 1
.6573	.7130	.6599	4	1 2 3
.6573	.7117	.6745	4	1 2 8
.6573	.6655	.6593	10	1 2 9
.6573	.6550	.4656	3	1 2 10
.6573	.7118	.6747	4	1 2 10 10
.6573	.9985	.6757	1	1 2 10 2
.6573	.9985	.6757	1	1 2 10 3
.6573	.7559	.7291	3	1 2 10 4
.6573	.7400	.7020	4	1 2 10 9
.6573	.7130	.6599	4	1 2 10 11
.6573	.7117	.6745	4	1 2 10 16
.6573	.6655	.6593	10	1 2 10 17
.6573	.6550	.4656	3	1 2 10 18
.6573	.7118	.6747	4	1 2 10 18 18
.6573	.9985	.6757	1	1 2 10 18 10
.6573	.9984	.6757	1	1 2 10 18 11
.6573	.7559	.7291	3	1 2 10 18 12
.6573	.7400	.7020	4	1 2 10 18 17
.6573	.7559	.7291	3	1 2 10 18 19
.6573	.7117	.6745	4	1 2 10 18 24
.6573	.6655	.6593	10	1 2 10 18 25
.6573	.6550	.4656	3	1 2 10 18 26
.6573	.7118	.6747	4	1 2 10 18 26 26
.6573	.9974	.6755	1	1 2 10 18 26 18
.6573	.9943	.7690	1	1 2 10 18 26 19
.6573	.7559	.7291	3	1 2 10 18 26 20
.6573	.7400	.7020	4	1 2 10 18 26 25
.6573	.6891	.6579	4	1 2 10 18 26 27
.6573	.7117	.6745	4	1 2 10 18 26 32
.6573	.6657	.5589	8	1 2 10 18 26 33
.6573	.6492	.4756	3	1 2 10 18 26 34
.6573	.6869	.6572	4	1 2 10 18 25 34 34
.6573	.7537	.7259	3	1 2 10 18 26 34 26
.6573	.7372	.7111	3	1 2 10 18 26 34 27
.6573	.7287	.7091	3	1 2 10 18 26 34 28
.6573	.7135	.6733	4	1 2 10 18 26 34 33
.6573	.6751	.6650	6	1 2 10 18 26 34 35
.6573	.6755	.6642	6	1 2 10 18 26 34 40
.6573	.6514	.5781	4	1 2 10 18 26 34 41
.6573	.6738	.6632	4	1 2 10 18 26 34 41 41
.6573	.7477	.6619	2	1 2 10 18 26 34 41 33
.6573	.7001	.6737	3	1 2 10 18 26 34 41 34
.6573	.6904	.6644	3	1 2 10 18 26 34 41 35
.6573	.6978	.6678	3	1 2 10 18 26 34 41 40
.6573	.6638	.6593	5	1 2 10 18 26 34 41 42
.6573	.6876	.6617	3	1 2 10 18 25 34 41 47
.6573	.6638	.6592	5	1 2 10 18 26 34 41 48
.6573	.6737	.6618	4	1 2 10 18 26 34 41 49
.6573	.6488	.5495	3	1 2 10 18 26 34 42

Figure 5.9 Trial Paths for a 7 by 7 Grid Problem With 10 Search Periods.

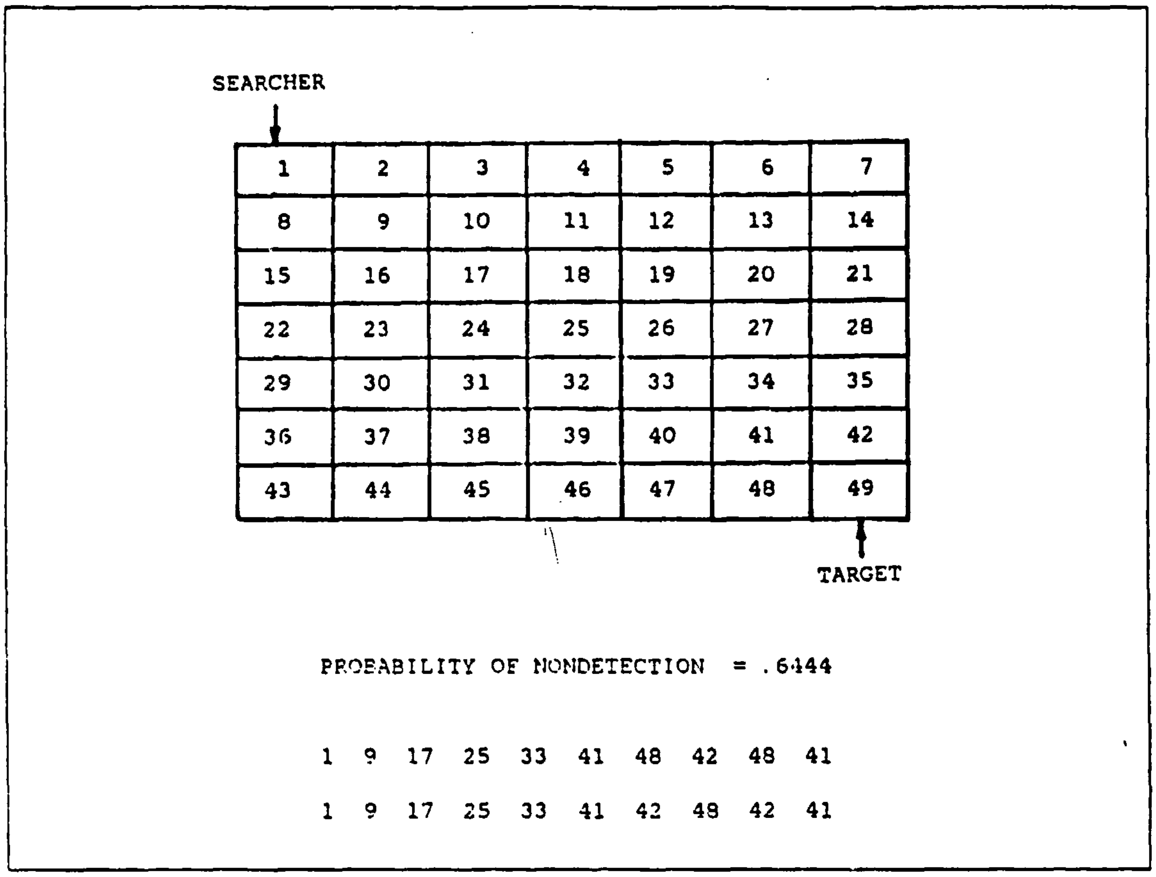


Figure 5.10 Optimal Integer Solutions for the 7 by 7 Grid With 10 Search Periods.

D. LARGER INTEGER APPLICATIONS

Using the same 7 by 7 grid as discussed in the previous section, the problem was run for successively longer intervals of search in order to get an idea of how rapidly computer run time increased with problem size. For each of these cases the target started in cell 49 and the searcher started in cell 1. A summary of run times and optimal search paths is shown in Table 2. Note that the optimal paths for each problem are essentially the same with the searcher going directly towards datum and then spreading out to cover the target's undetected probability mass. We can almost detect something that resembles a systematic search, especially for the 12 time period solution. Here it looks like the searcher is begining to expand his area of coverage by moving outwards from datum. Unfortunately, the 13 time period problem was not solvable within 1 hour of run time on the IBM 3033 mainframe, therefore we are unable to see what happens with more search periods. Thus we are unable to comment on the validity of systematic search. Run times as a function of problem size for this 49 cell grid are illustrated on the graph in Figure 5.11. We can see how rapidly the run time increases as the number of search periods increases from 11 to 12. This probably results more from the increase in the number of possible searcher paths than from the increase in the number of arcs in the network.

E. SUMMARY

The overiding consideration in all of these applications is how much larger can we go? While the divisible search effort algorithm seems to be efficient and capable of handling very large problem sizes, the run times for the integer solution appear to grow rapidly as problem size increases. With this in mind, a better question might be how large do we *need* to go? Here the major consideration is the purpose for which the problem is being solved. If we are interested only in learning about optimal approaches to various search problems we may be willing to tolerate the long run times associated with larger integer problems. Yet for employment of the procedure in real world situations long run times are unacceptable. This may dictate the use of other techniques for solving the integer problem. One possible method is to model the problem using an infinite time horizon with discounting, where early detections are more heavily weighted than later detections. With this model, iarger integer problems might be solvable, however the choice of appropriate discount factors will be difficult. Still, this technique is worthy of consideration.

Another consideration is the *type* of search problem to be solved. Thus far the integer solutions we have looked at constitute only a specific type of datum search in which the target starts at one corner of the grid and the searcher at the opposite corner. What happens if instead the target begins the problem in the center of the grid? This very problem was run using the branch-and bound procedure for 5 by 5 and 7 by 7 grids with 10 time periods of search. In each case, the algorithm did not achieve an optimal solution after one hour of computer run time. This was surprising especially after the "fast" run times associated with the previous 5 by 5 and 7 by 7 datum searches. A possible explanation for this is that for some problems the relaxation on divisiblity of search effort within the subprogram results in weak lower bounds. Recall the 3 by 3 case with 10 time periods of search as discussed in Chapter IV. For this problem some 3940 trial paths were generated. Yet for the 7 by 7 application of Section 3 above only 945 trial paths were considered despite the fact that the later case is significantly larger. Close inspection of the divisible search effort solutions to both

TABLE 2

RUN TIMES AND PATHS FOR LARGE INTEGER PROBLEMS

GRID SIZE	TIME PERIODS	NUMBER OF ARCS	RUN TIME (MIN)	NUMBER OF TRIAL PATHS	OPTIMAL PATHS	PND
7x7	10	3249	4:40	945	1 9 17 25 33 41 48 42 48 41	.6444
					1 9 17 25 33 41 42 48 42 41	.6444
7x7	11	3610	7:32	1228	1 9 17 25 33 41 48 42 48 42 41	.6064
					1 9 17 25 33 41 42 48 42 48 41	.6064
7x7	12	3971	46:12	8046	1 9 17 25 33 41 48 42 48 42 34 41	.5767
					1 9 17 25 33 41 42 48 42 48 40 41	.5767
7x7	13	4332	>HR	-	-	-

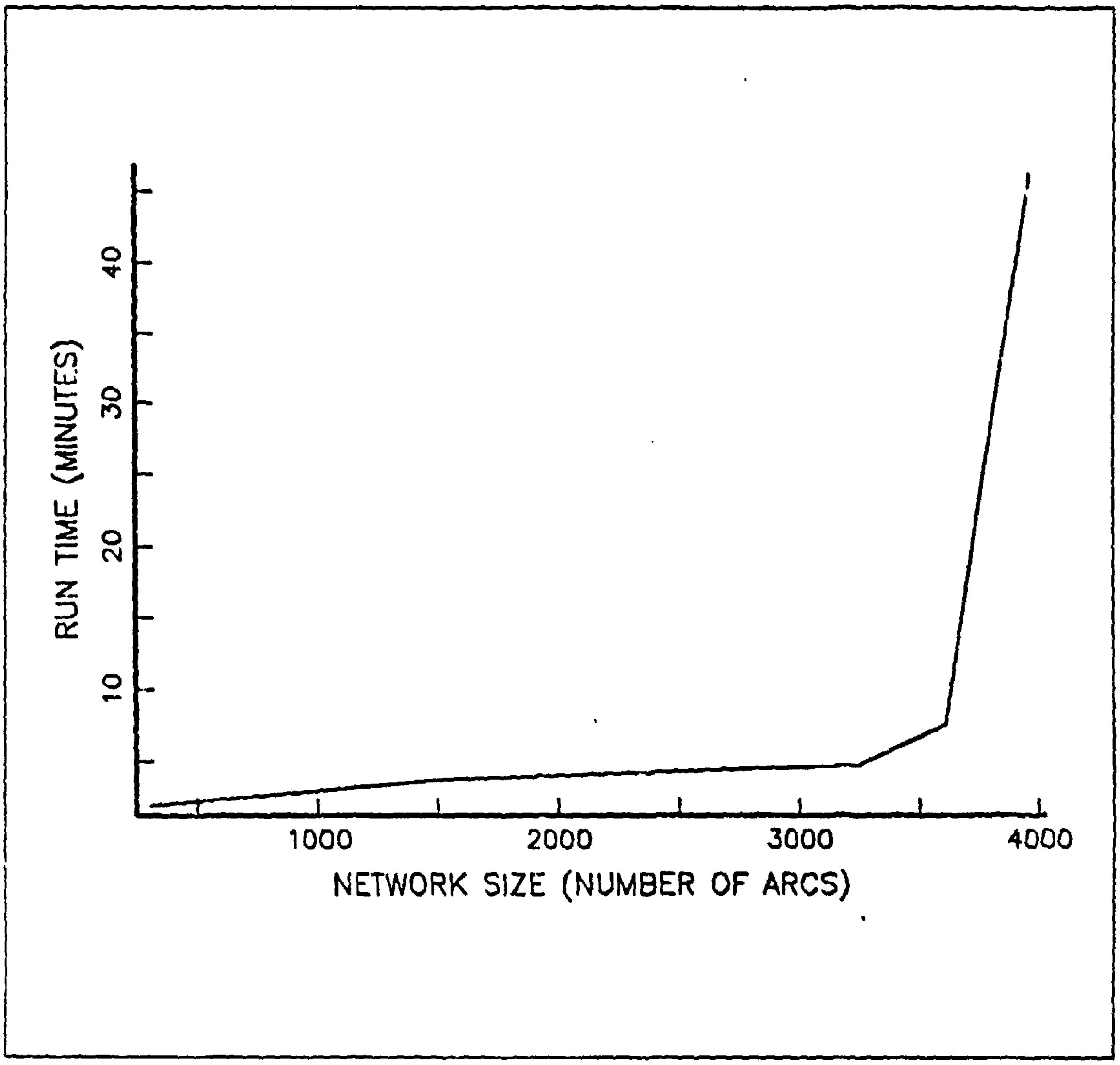


Figure 5.11 Graph Displaying Run Times for Integer Problems.

problems show that for the 7 by 7 case the search effort does not fractionate until time period 6, resulting in a near-integer solution. Conversely in the 3 by 3 case the search effort is divided immediately. It appears that the divisible search effort subroutine is better in calculating lower bounds for the 7 by 7 case than the 3 by 3 case. This consideration may prove very important when attempting to solve other search problems of various geometries.

VI. CONCLUSIONS

A. SUMMARY OF HIGHLIGHTS

We have seen successful implementation of the branch-and-bound procedure as proposed by Professors Eagle and Yee. After many applications of this technique and the divisible search effort subprogram to various datum searches, there are several key items of significance worth noting:

- The divisible search procedure ran quickly and efficiently on all scenarios tested. Additionally, the relatively short run time required to solve the 15 by 15 cell case in Chapter V, suggested that much larger problems could be solved with this algorithm.
- The computer run times required to find integer solutions grew rapidly with problem size. It appeared that simply increasing the number of time periods for the problem had a more significant effect on run time than increasing the size of the grid. This observation is substantiated by noting the an increase of approximately 1 minute in going from a 5 by 5 grid to a 7 by 7 grid both with 10 search periods (see Table 1). Comparatively an increase of almost 3 minutes was observed in going from a 10 time period 7 by 7 case to the same problem with 11 time periods. This condition may prevent the branch-and-bound procedure from being implemented in large grid problems, for there is a desireable relationship between grid size and solvable time horizon. With a larger problems more search periods are required in order to allow the searcher adequate time to span the grid. Therefore, if time horizons are most limiting, only smaller grids may be considered.
- Lower bounds calculated in the subprogram are much stronger when the divisible search effort solution closely parallels the integer solution. This means that fewer trial paths are considered when the lower bound is strong resulting in faster run times. This result may prove limiting in the types of geometries that may be solved by the branch-and bound procedure, however more testing is required to substantiate this conclusion.
- Although we would like to comment on the validity of systematic search there are not enough test cases to do so. However, there does seem to be some kind of systematic approach to the datum searches that were considered. Each has the searcher speeding off towards datum and then hopping back and forth across cells adjacent to the datum cell. In one case we did actually observe what appeared to be a searcher expanding his area of coverage, but not enough time periods were covered in order to make a valid conclusion.

B. PROPOSED REAL WORLD APPLICATIONS

Of the two cases considered the divisible search effort algorithm seems to be the most promising as far as real world application. Of course modifications to the procedure would be necessary, but some possible applications include:

- Sonobouy placement by aircraft: In this case each sonobouy may be considered as a separate searcher. For each time period the aircraft has a finite number of resources that he must distribute over the area of uncertainty. Because of his speed advantage over a submarine target, he may move almost "instantaneously" in order to spread this effort.
- Mine placement: Here again mines could be considered as "searchers".
- Large search parties: This might be a group of men patrolling a large area as suggested by Stewart [Ref. 3: p.129], or perhaps a collection of a aircraft sweeping over an area for a downed pilot.
- A single aircraft: Although this case involves a single searcher, the aircraft's speed advantage allows him to cover more than one cell in a given time period.

As far as integer applications, if faster run times are available, this procedure could be used for any case involving a single unit as the searcher. Of course for some applications the problem sizes as discussed within this report may be adequate. Even if it is not possible to achieve lower run times, a hybrid combination of both the divisible and integer algorithms might be feasible. Consider a user selecting various "best guess" integer solutions for evaluation on a console that returns the value of the probability of nondetection for each "best guess". The divisible search algorithm might be called to show the optimal allocation of search effort for each time period. Seeing this, the user may generate or even modify his "best guess" path before submitting it for evaluation. In the background of all this is the integer solution slowly churning away, eventually to be printed out on the console. Also as Stewart noted [Ref. 3: p.135], the first full solution generated by the branch-and-bound procedure when at the bottom of the tree ($t = T$), is very close to optimality and might be suitable as a heuristic solution.

C. UNANSWERED QUESTIONS

This paper has merely scratched the surface of the complete investigation for the integer and divisible search procedures of Professors Eagle and Yee, for there are still many areas of interest with regard to application and implementation.

From the implementation aspect there are undoubtablely several areas for improvement, especially concerning the line search. What is the best line search technique for this application and how much accuracy in the line search is required in order to find good lower bounds for use in the branch-and-bound procedure? Quite possibly the Goldstein-Armijo conditions [Ref. 7: Chap.2] could be implemented to find an acceptable stopping point for the line search. These questions have not been thoroughly investigated. Also, there is the question of when to terminate the search for the lower bound and again how much accuracy is enough? Additionally the branching discipline for the integer program in this report is determined by the order in which the cells are listed in the input file. Some better method is necessary to implement good branching rules that may help reduce the overall run times. And finally for all applications presented, the search effectiveness parameter α_{ij} has essentially been ignored. Recall that it was set equal to 1 for simplicity; the effects of varying this parameter are still unknown.

Aside from implementation there are various scenarios yet to be considered. What of the cases involving the search of an area where the target is initially uniformly distributed? Or how about the search for a transiting target? And what happens if the target is at high speed versus low speed? The possible scenarios are endless. The big question here is just what types of problems can be solved with the branch-and-bound procedure? We've already mentioned the effects of grid size and suitable time horizons. Will we be able to solve a large enough problem to investigate these cases? But also there is the concern of the strength of the lower bounds for certain geometries. All of these things are yet to be completely understood.

Still the most crucial question is what can we learn about optimal search paths? Will we be able to use this branch-and-bound algorithm, or for that matter any procedure that solves the integer problem, to help substantiate that systematic search is the best method? Or instead could the algorithm be used to help develop heuristics for the different classes of search problems? In order to answer these questions fully more investigation is clearly warranted.

APPENDIX A
DERIVATION OF COMPUTATIONAL FORMULAS

1. CALCULATING THE PROBABILITY OF NONDETECTION

Recall from Chapter II that given a set of flows the probability of nondetection could be calculated from equation A.1 as follows:

$$Q = \sum_{\omega} p_{\omega} \exp\{-X(\omega(1),1) - \sum_{t=2}^{T} \sum_{i \in C_{\omega(t)}} \alpha_{i,\omega(t)}\, x(i,\omega(t),t\text{-}1)\} \qquad \text{(eqn A.1)}$$

This formula is useful for demonstrating the convexity of the objective function yet is not very practical for computational purposes because all possible target paths must be completely enumerated. A better way to make this calculation is to exploit the Markovian nature of the target's motion. We can do this by using a matrix to keep track of the target probability mass within each cell. Then by post-multiplying this matrix by a "nondetection probability matrix" and a target transition matrix, the target's probability distribution may be updated iteratively for each time period. At the end of T time periods, the overall probability of nondetection may be found by summing the remaining mass among all cells. This iterative procedure is illustrated below in more detail.

Consider a general N cell, T time period problem with the target starting in cell N and the searcher initially in cell 1. The target's probability distribution can be given by the 1 x N matrix $\underline{P}$ where:

$$\underline{P} = [0, 0, \ldots, 0, 1] \qquad \text{(eqn A.2)}$$

(In general let $\underline{P}$ represent any 1 x N matrix showing the target's mass distribution). In time period 1 the searcher conducts a search. For each cell the probability of nondetection given that the target is within the cell is found by $\exp\{-X(i,1)\}$ where $X(i,1)$ is the amount of search effort in cell i during time period 1.

In order to calculate the target mass remaining after the search in time period 1, each entry in $\underline{P}$ must be multiplied by its associated probability of nondetection as given in above. To retain the proper shape of the resulting matrix, these probabilities are placed along the diagonal of an N x N matrix and then pre-multiplied by $\mathbf{P}$ as shown below:

$$\mathbf{P} = \begin{bmatrix} 0, 0, \ldots, 0, 1 \end{bmatrix} \begin{bmatrix} \exp\{-X(1,1)\} & & 0 \\ & \ddots & \\ 0 & & \exp\{-X(N,1)\} \end{bmatrix} \qquad \text{(eqn A.3)}$$

The result is a 1 x N matrix showing the target mass remaining in each cell following the search in time period 1. This mass must now transition into time period 2 by post-multiplying $\mathbf{P}$ by the N x N Markov transition matrix $\mathbf{\Gamma}$. As before, the result is a 1 x N matrix of the target's mass distribution within each cell, but this time at the start of time period 2. To account for the next search, the procedure is repeated except now the probabilities of nondetection are computed using the flows as shown in equation A.4 below.

$$\exp\{-\sum_{i \in C_j} \alpha_{ij}\, x(i,j,1)\} \qquad \text{(eqn A.4)}$$

As in time period 1 these probabilities can be arranged along the diagonal of an N x N matrix and then used to update $\mathbf{P}$ as follows:

$$\mathbf{P} = \begin{bmatrix} 0, 0, \ldots, 0, 1 \end{bmatrix} \begin{bmatrix} \exp\{-X(1,1)\} & & 0 \\ & \ddots & \\ 0 & & \exp\{-X(N,1)\} \end{bmatrix} \mathbf{\Gamma}$$

$$\begin{bmatrix} \exp\{-\sum_{i \in C_1} \alpha_{i1}\, x(i,1,1)\} & & 0 \\ & \ddots & \\ 0 & & \exp\{-\sum_{i \in C_N} \alpha_{iN}\, x(i,N,1)\} \end{bmatrix} \qquad \text{(eqn A.5)}$$

These same procedures are repeated for all T time periods to yield the final $\underline{P}$ matrix:

$$\underline{P} = [0, 0, \ldots, 0, 1] \begin{bmatrix} \exp\{-\mathbf{X}(1,1)\} & & & 0 \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ 0 & & & \exp\{-\mathbf{X}(N,1)\} \end{bmatrix} \Gamma$$

$$\begin{bmatrix} \exp\{-\sum_{i \in C_1} \alpha_{i1} \, x(i,1,1)\} & & & 0 \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ 0 & & & \exp\{-\sum_{i \in C_N} \alpha_{iN} \, x(i,N,1)\} \end{bmatrix} \Gamma$$

$$\vdots$$

$$\begin{bmatrix} \exp\{-\sum_{i \in C_1} \alpha_{i1} \, x(i,1,T-1)\} & & & 0 \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ 0 & & & \exp\{-\sum_{i \in C_N} \alpha_{iN} \, x(i,N,T-1)\} \end{bmatrix} \quad \text{(eqn A.6)}$$

Once this matrix is found, the overall probability of nondetection is calculated by summing all remaining target mass.

2. CALCULATING PARTIAL DERIVATIVES

Consider a single variable in the previous problem, $x(\hat{i}, \hat{j}, \hat{t})$. We would like to calculate the partial derivative of $\underline{P}$ with respect to this variable. Rewriting the $\underline{P}$ matrix from Section 1 to show the $x(\hat{i}, \hat{j}, \hat{t})$ term we have:

$$\underline{P} = \left[0, 0, \ldots, 0, 1\right] \begin{bmatrix} \exp\{-\mathbf{X}(1,1)\} & & 0 \\ & \ddots & \\ 0 & & \exp\{-\mathbf{X}(N,1)\} \end{bmatrix} \Gamma$$

$$\begin{bmatrix} \exp\{-\sum_{i \in C_1} \alpha_{i1}\, x(i,1,1)\} & & 0 \\ & \ddots & \\ 0 & & \exp\{-\sum_{i \in C_N} \alpha_{iN}\, x(i,N,1)\} \end{bmatrix} \Gamma$$

$$\vdots$$

$$\begin{bmatrix} \exp\{-\sum_{i \in C_1} \alpha_{i1}\, x(i,1,\hat{t}) & & & 0 \\ & \ddots & & \\ & & \exp\{-\alpha_{\hat{i}\hat{j}}\, x(\hat{i},\hat{j},\hat{t}) - \sum_{i \in \{C_{\hat{j}} - \hat{i}\}} \alpha_{i\hat{j}}\, x(i,\hat{j},\hat{t})\} & \\ & & \ddots & \\ 0 & & & \exp\{-\sum_{i \in C_N} \alpha_{iN}\, x(i,N,\hat{t})\} \end{bmatrix} \Gamma$$

$$\vdots$$

$$\begin{bmatrix} \exp\{-\sum_{i \in C_1} \alpha_{i1}\, x(i,1,T-1)\} & & 0 \\ & \ddots & \\ 0 & & \exp\{-\sum_{i \in C_N} \alpha_{iN}\, x(i,N,T-1)\} \end{bmatrix} \qquad \text{(eqn A.7)}$$

As suggested by Brown [Ref. 1: pp.1281-2], this may be rewritten again as follows:

$$\underline{R}\left[j,\hat{t}+1\right]\begin{bmatrix}\exp\{-\sum_{i\in C_1}\alpha_{i1}\,x(i,1,\hat{t})\} & & & 0\\ & \cdot & & \\ & & \cdot & \\ 0 & & & \exp\{-\sum_{i\in C_N}\alpha_{iN}\,x(i,N,\hat{t})\}\end{bmatrix}\underline{S}\left[j,\hat{t}+1\right] \quad \text{(eqn A.8)}$$

where $\underline{R}\left[j,\hat{t}+1\right]$ is a 1 x N matrix that shall be referred to as the "reach" matrix, and $\underline{S}\left[j,\hat{t}+1\right]$ is an N x N matrix referred to as the "survival" matrix. The reason for these names will become apparent shortly. Using this matrix notation makes calculation of the partial derivatives fairly easy, for the $\underline{R}$ and $\underline{S}$ matrices contain no terms that include the variable $x(\hat{i},\hat{j},\hat{t})$ and therefore may be treated as constants. Finally the partial derivative of $\underline{P}$ with respect to $x(\hat{i},\hat{j},\hat{t})$ is calculated:

$$\frac{\partial \underline{P}}{\partial x(\hat{i},\hat{j},\hat{t})} = R(\hat{j},\hat{t}+1)\,(\exp\{-\sum_{i\in C_{\hat{j}}}\alpha_{i\hat{j}}\,x(i,\hat{j},\hat{t})\})\,(-\alpha_{\hat{i}\hat{j}})\,S(\hat{j},\hat{t}+1) \qquad \text{(eqn A.9)}$$

Now $R(\hat{j},\hat{t}+1)$ is a real number giving the probability that the target reaches cell $\hat{j}$ in time period $\hat{t}+1$ and $S(\hat{j},\hat{t}+1)$ is a real number representing the probability of target survival to time T given that there was no detection in cell $\hat{j}$ for time $\hat{t}+1$. Note that because of the diagonal matrices involved, all other terms in $\underline{R}$ and $\underline{S}$ go to 0. With this formula, partial derivatives are easily calculated for all flow variables. Additionally it is important to note that $\underline{R}\left[j,t\right]$ may be calculated iteratively as shown:

$$\underline{R}\left[j,t+1\right] = \underline{R}\left[j,t\right]\begin{bmatrix}\exp\{-\sum_{i\in C_1}\alpha_{i1}\,x(i,j,t-1)\} & & & 0\\ & \cdot & & \\ & & \cdot & \\ 0 & & & \exp\{-\sum_{i\in C_N}\alpha_{iN}\,x(i,N,t-1)\}\end{bmatrix}\Gamma \qquad \text{(eqn A.10)}$$

Similarly, survival matrices $\underline{S}[j,t]$ may also be calculated iteratively as shown below:

$$\underline{S}[j,t] = \boldsymbol{\Gamma} \begin{bmatrix} \exp\{-\sum_{i \in C_1} \alpha_{i1}\, x(i,j,t) & & & 0 \\ & \cdot & & \\ & & \cdot & \\ & & & \cdot \\ 0 & & & \exp\{-\sum_{i \in C_N} \alpha_{iN}\, x(i,N,t)\} \end{bmatrix} \underline{S}[j,t+1] \qquad \text{(eqn A.11)}$$

The recursion begins with $\underline{S}[j,T]$ being a column vector of ones and $\underline{R}[j,1]$ being a row vector of the target's initial distribution over the cells.

APPENDIX B
DIVISIBLE SEARCH EFFORT PROGRAM LISTING

1. SOME DETAILS ON PROGRAMMING METHODS

As previously mentioned the divisible search algorithm was coded in Fortran and run in an IBM 3033 mainframe computer. The program is written to accommodate changes in problem size very easily. This is done by the use of "Parameter" statements to control array sizes and stopping points for iterative computations. Extensive use of subroutines allows for efficiency as well as ease of understanding. Input is from an data file which is necessary in order to handle the large amount of imformation that must be used within the algorithm such as transition probabilities, adjacent cell numbers, etc. To help minimize the overall storage requirements of this data, adjacency lists and entry point arrays are used extensively to represent the arcs and flows within the network. These will not be discussed here but instead are adqeuately described by Reference 6. A program listing is provided in Section 2, for which the parameters are set up for a 25 cell problem with 10 search periods.

2. PROGRAM LISTING

```
      PROGRAM MAIN
************************************************************************
*      PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                    *
* PURPOSE:                                                             *
*    THIS IS THE CONTROLLING PROGRAM FOR THE CONSTRAINED SEARCH        *
* ALGORITHM WITH DIVISIBLE SEARCH EFFORT. IT SERVES MERELY TO CALL     *
* MAJOR SUBROUTINES THAT IMPLEMENT THE PROCEDURE.                      *
*                                                                      *
* KEY VARIABLES:                                                       *
*       A: A MATRIX OF SEARCH EFFECTIVENESS PARAMETERS FOR EACH        *
*          ARC IN THE NETWORK. LISTED IN ADJACENCY LIST FORM.          *
*     ADD: A MATRIX SIMILAR TO THE ADJACENCY LIST BUT INSTEAD OF GIVING *
*          THE HEADS OF ARCS INCIDENT TO CELLS LISTED IN THE ENTRY POINT *
*          THIS MATRIX GIVES THE TAILS OF ALL ARCS THAT FLOW INTO THE  *
*          CELL LISTED IN EP.                                          *
*     ADJ: A MATRIX THAT GIVES THE HEADS OF ALL ARCS IN ADJACENCY LIST *
*          FORMAT.                                                     *
*DELMIN: THE USER DEFINED INTERVAL OF ACCURACY REQUIRED FOR THE        *
*          LOWER BOUND. THIS ALSO SPECIFIES THE STOPPING CRITERIA      *
* DELTA: THE CHANGE IN PROBABILITY OF NONDETECTION PREDICTED BY THE    *
*          FIRST ORDER TAYLOR APPROXIMATION IN GOING FROM SOLUTION     *
*          X1 TO SOLUTION X2.                                          *
*      EP: THE ENTRY POINT ARRAY FOR THE ADJACENCY LISTS.              *
* EPLEN: A PARAMETER THAT IS USED TO SET THE DIMENSION OF THE EP ARRAY *
*  FRAC: A MATRIX OF DIMENSIONS NCELLS BY TMAX IN WHICH ENTRY,         *
*          FRAC(I,T) GIVES THE FRACTION OF CELL I SEARCHED IN TIME T.  *
*  GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC      *
*          IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE  *
*          OBJECTIVE FUNCTION WITH RESPECT TO ARC J FROM THE ADJ. LIST. *
*LENGTH: A PARAMETER USED TO SET DIMENSIONS OF ALL ADJACENY LISTS.     *
*NCELLS: A PARAMETER SPECIFYING THE NUMBER OF CELLS IN THE SQUARE GRID *
*  NEXT: AN ARRAY USED TO KEEP TRACK OF THE SHORTEST PATH.             *
```

```
*   PND: THE PROBABILITY OF NONDETECTION                              *
*  PND1: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*        BY X1 (THE START POINT).                                    *
*  PND2: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*        BY X2 (THE EXTREME POINT).                                  *
*  PND3: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*        BY X1 (THE MIDPOINT IN THE QUADRATIC LINE SEARCH).          *
*  PND4: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*        BY X4 (THE MINIMIZING POINT FROM THE LINE SEARCH).           *
*     R: THE MATRIX OF DIMENSION NCELLS BY TMAX OF REACH PROBS.       *
*     S: THE MATRIX OF DIMENSION NCELLS BY TMAX OF SURVIVE PROBS.     *
* START: THE SEARCHER'S INITIAL FEASIBLE SOLUTION                     *
*     T: AN INTEGER REPRESENTING PROBLEM TIME.                        *
*TGSTRT: A MATRIX GIVING THE TARGET STARTING DISTRIBUTION ON THE GRID. *
* TGTDN: THE CURRENT TARGET DENSITY.                                  *
*TGTDNF: THE FUTURE TARGET DENSITY AFTER ONE MARKOV TRANSITION.       *
*TGTDNP: THE PAST TARGET DENSITY ONE MARKOV TRANSITION BACKWARDS.     *
* THETA: THE FRACTION OF THE DISTANCE FROM X1 TO X2 THAT MINIMIZES    *
*        THE OBJECTIVE FUNCTION.                                      *
*  TMAX: THE TOTAL NUMBER OF SEARCH PERIODS.                          *
* TRANS: A MATRIX GIVING THE MARKOV TRANSITION PROBABILITIES FOR EACH *
*        ARC LISTED IN ADJACENCY LIST FORMAT.                         *
* TRIAL: A DUMMY VARIABLE USED TO KEEP TRACK OF VOC DURING THE SHORT- *
*        TEST PATH ROUTINE.                                           *
*   VOC: THE VALUE OF CONTINUING FOR EACH NODE ON THE SHORTEST PATH.  *
*     X: A SET OF ANY FEASIBLE FLOWS. (ALL FLOW VARIABLES ARE GIVEN   *
*        IN ADJACENCY LIST FORMAT.)                                   *
*    X1: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE START POINT.     *
*    X2: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT.   *
*    X3: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE MIDPOINT IN THE  *
*        QUADRATIC LINE SEARCH.                                       *
*    X4: THE SET OF FEASIBLE FLOWS ASSOCIATED WITH THE MINIMIZING POINT*
*        FROM THE QUADRATIC LINE SEARCH.                              *
*    XO: AN ARRAY GIVING THE THE AMOUNT OF SEARCH EFFORT IN EACH CELL. *
*                                                                     *
* REFERENCE:                                                          *
*   CON FORTRAN  WRITTEN BY PROFESSOR JAMES EAGLE AT THE NAVAL PG     *
*  SCHOOL IN MONTEREY, CALIFORNIA, TO SOLVE THE DIVISIBLE PROBLEM.    *
***********************************************************************

*                                           ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),START(TMAX),ADD(LENGTH)
      REAL TRANS(LENGTH),A(LENGTH),TGSTRT(NCELLS),XO(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*
      CALL INPUT(START,DELMIN)
      CALL LOWBND(START,DELMIN)
*
      STOP
      END


      SUBROUTINE BOUND(X1,X2,GRAD,DELTA)
***********************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                    *
* PURPOSE:                                                            *
*   THIS PROGRAM COMPUTES THE DELTA FOR USE IN CALCULATING THE LOWER  *
* BOUND ASSOCIATED WITH EACH FRANK-WOLFE ITERATION. THIS DELTA IS THE *
* CHANGE IN THE PROBABILITY OF NONDETECTION ACHIEVED BY GOING FROM X1 *
* TO X2 AND IS CALCULATED BY THE FIRST ORDER TAYLOR APPROXIMATION.    *
*                                                                     *
* INPUT:                                                              *
*   X1: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE START POINT       *
*   X2: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT     *
*   GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC    *
*         IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE  *
*         OBJECTIVE FUNCTION WITH RESPECT TO ARC J.                   *
```

```
* OUTPUT:                                                               *
*   DELTA: THE CHANGE IN PROBABILITY OF NONDETECTION PREDICTED BY THE  *
*          FIRST ORDER TAYLOR APPROXIMATION IN GOING FROM SOLUTION     *
*          X1 TO SOLUTION X2.                                          *
************************************************************************
*                                         ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH)
      REAL X1(LENGTH,TMAX),X2(LENGTH,TMAX),GRAD(LENGTH,TMAX),XO(NCELLS),
     1TGSTRT(NCELLS),TRANS(LENGTH),A(LENGTH)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
      DELTA=0
      DO 10 T=1,TMAX-1
            DO 10 J=1,EP(NCELLS+1)-1
               DELTA=DELTA+GRAD(J,T)*(X2(J,T)-X1(J,T))
   10 CONTINUE
      RETURN
      END


      SUBROUTINE FRACT(X,FRAC)
************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                     *
* PURPOSE:                                                             *
*   THIS PROGRAM CALCULATES THE FRACTION OF CELL I SEARCHED IN TIME    *
* PERIOD T. THIS FRACTION IS SIMPLY THE SUM OF OVER ALL ADJACENT CELLS *
* OF THE PRODUCT OF FLOW EFFORT AND SEARCH EFFECTIVENESS.              *
*                                                                      *
* INPUT:                                                               *
*   X:  A SET OF FEASIBLE FLOWS                                        *
*                                                                      *
* OUTPUT:                                                              *
*  FRAC: A MATRIX OF DIMENSIONS NCELLS BY TMAX IN WHICH ENTRY,         *
* FRAC(I,T) GIVES THE FRACTION OF CELL I SEARCHED IN TIME PERIOD T.    *
************************************************************************
*
*                                         ... DECLARATIONS
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH)
      REAL A(LENGTH),FRAC(NCELLS,TMAX),TRANS(LENGTH),X(LENGTH,TMAX),
     1XO(NCELLS),TGSTRT(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*                                         ... COMPUTE FRACTION OF CELL I
*                                             SEARCHED IN TIME PERIOD T
*                                             BY SUMMING FLOWS FROM ALL
*                                             ADJACENT CELLS
      DO 20 I=1,NCELLS
                   FRAC(I,1)=XO(I)
         DO 15 T=2,TMAX
                   FRAC(I,T)=0
              DO 10 J=EP(I),EP(I+1)-1
                   FRAC(I,T)=FRAC(I,T)+A(ADD(J))*X(ADD(J),T-1)
   10         CONTINUE


   15    CONTINUE
   20 CONTINUE
      RETURN
      END


      SUBROUTINE GRADF(FRAC,GRAD)
************************************************************************
```

```
*      PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                       *
* PURPOSE:                                                                *
*   THIS PROGRAM CALCULATES THE PARTIAL DERIVATIVES OF THE OBJECTIVE      *
* FUCTION WITH RESPECT TO EACH ARC IN THE NETWORK.                        *
*                                                                         *
* INPUT:                                                                  *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH     *
*         TIME PERIOD.                                                    *
*                                                                         *
* OUTPUT:                                                                 *
*   GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC        *
*         IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE      *
*         OBJECTIVE FUNCTION WITH RESPECT TO ARC J.                       *
***************************************************************************
*
*                                           ... DECLARE / INITIALIZE
*
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH)
      REAL A(LENGTH),R(NCELLS,TMAX),S(NCELLS,TMAX),FRAC(NCELLS,TMAX),
     1GRAD(LENGTH,TMAX),TRANS(LENGTH),X(LENGTH,TMAX),XO(NCELLS),
     2TGSTRT(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*                                           ...DETERMINE REACH AND SURVIVE
*                                              PROBABILITIES
      CALL REACH(FRAC,R)
      CALL SURVIV(FRAC,S)
*                                           ... CALCULATE PARTIAL DERIVATIVES
*                                               FOR EACH FLOW X(.,T)
      DO 10 T=1,TMAX-1
          DO 10 I=1,NCELLS
               DUMMY=-R(I,T+1)*EXP(-FRAC(I,T+1))*S(I,T+1)
              DO 10 J=EP(I),EP(I+1)-1
               GRAD(ADD(J),T)=DUMMY*A(ADD(J))
   10 CONTINUE
      RETURN
      END


      SUBROUTINE INPUT(START,DELMIN)
***************************************************************************
*      PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                       *
* REFERENCE:                                                              *
*   THIS PROGRAM READS IN DATA FROM AN INPUT FILE FOR USE FOR THE         *
* CONSTRAINED SEARCH ALGORITHM.                                           *
***************************************************************************
*                                           ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER NADJ,EP(EPLEN),ADJ(LENGTH),START(TMAX),T,ADD(LENGTH)
      REAL TRANS(LENGTH),A(LENGTH),TGSTRT(NCELLS),XO(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*                                           ... READ DESIRED ACCURACY OF
*                                               LOWBOUND, DELMIN
      READ(01,*) DELMIN
*
*                                           ... READ IN ADJACENT CELL NUMBERS
*                                               IN ADJACENCY LIST FORM WITH
*                                               ENTRY POINT ARRAY, EP(.), AND
*                                               HEADARRAY, ADJ(.).
      T=1
      DO 5 I=1,NCELLS
          EP(I)=T
          READ(01,*) DUMMY,NADJ,(ADJ(J), J=T,T+NADJ-1)
          T=T+NADJ
```

```
    5 CONTINUE
      EP(NCELLS+1)= T
      ADJ(T)=0
*                                          ... GENERATE ADDRESS ARRAY ADD(.)
*                                              FOR EACH CELL THE ADD ARRAY
*                                              GIVES A LIST OF ENTRY POINT
*                                              POSITIONS IN THE ADJACENCY
*                                              LIST OF ALL ARCS THAT FLOW
*                                              INTO THE CELL
      L=1
      DO 8 I=1,NCELLS
         DO 8 K=1,LENGTH
              IF(ADJ(K).EQ.I) THEN
                   ADD(L)=K
                   L=L+1
              END IF
    8 CONTINUE
      ADD(L)=0
*                                          ... READ IN TARGET TRANSITION
*                                              PROBABILITIES TRANS (.) IN
*                                              ADJACENCY LIST FORM
      DO 10 I=1,NCELLS
         READ(01,*) DUMMY,(TRANS(J), J=EP(I),EP(I+1)-1)
   10 CONTINUE
      TRANS(EP(NCELLS+1))=0
*     PRINT * ,TRANS(EP(NCELLS+1)-1)
*                                          ... READ IN SEARCH EFFECTIVENESS
*                                              A(.), IN ADJACENCY LIST FORM
      DO 20 I=1,NCELLS
         READ(01,*) DUMMY,(A(J), J=EP(I),EP(I+1)-1)
   20 CONTINUE
      A(EP(NCELLS+1))=0
*                                          ... READ IN STARTING SOLUTION,
*                                              START(.)
*
      READ(01,*) (START(T), T=1,TMAX)
*
*                                          ... READ IN INITIAL TARGET
*                                              DISTRIBUTION, TGSTRT(.)
      DO 30 I=1,NCELLS
   30    READ(01,*) DUMMY,TGSTRT(I)
*
   11 RETURN
      END


         SUBROUTINE LOWBND(START,DELMIN)
*************************************************************************
*     PROGRAMMER: FRANK CALDWELL     DATE: SEP 87                        *
* PURPOSE:                                                               *
*   THIS SUBROUTINE REPRESENTS THE SUBSTANTIAL PART OF THE DIVISIBLE     *
* EFFORT PROGRAM. IT CONTROLS THE ITERATIVE SEQUENCE OF THE SOLUTION     *
* TECHNIQUE BY CALLING VARIOUS SUBROUTINES TO LINEARIZE THE OBJECTIVE    *
* FUNCTION, FIND THE SHORTEST PATH, ETC.                                 *
*                                                                        *
* INPUTS:                                                                *
*   START: THE SEARCHER'S INITIAL FEASIBLE SOLUTION                      *
*   DELMIN: THE USER DEFINED INTERVAL OF ACCURACY REQUIRED FOR THE       *
*           LOWER BOUND. THIS ALSO SPECIFIES THE STOPPING CRITERIA       *
* OUTPUTS:                                                               *
*   SOLUTIONS TO THE DIVISIBLE SEARCH EFFORT PROBLEM. THESE ARE PRO-     *
* VIDED THROUGH THE SUBROUTINE OUTPUT.                                   *
*************************************************************************
*
*                                          ... DECLARE / INITIALIZE
```

```
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER  EP(EPLEN),ADJ(LENGTH),START(TMAX),T,ADD(LENGTH)
      REAL X1(LENGTH,TMAX),XO(NCELLS),FRAC(NCELLS,TMAX),
     1TGSTRT(NCELLS),S(NCELLS,TMAX),A(LENGTH),GRAD(LENGTH,TMAX),
     2TRANS(LENGTH),X2(LENGTH,TMAX),X4(LENGTH,TMAX)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*
*                                         ... CALCULATE FLOWS FOR THE
*                                             INITIAL FEASIBLE SOLUTION
      PLOW=0
      DO 10 I=1,NCELLS
            XO(I)=0
   10 CONTINUE
      XO(START(1))=1
      INDEX=EP(START(1))
      DO 12 T=1,TMAX-1
         DO 11 J=1,EP(NCELLS+1)-1
                 X1(J,T)=0
   11    CONTINUE
         X1(INDEX,T)=1
         INDEX=EP(ADJ(INDEX))
   12 CONTINUE
****************** CALCULATIONS OF THE LOWER BOUND *******************
*                                         ... FIND PND1, INITIAL NON-
*                                             DETECTTION PROBABILITY
      CALL PNDET(X1,FRAC,PND1)
*    WRITE(11,'(1X,A16,1X,F5.4)') 'INITIAL PBEST IS',PND1
   15 CALL GRADF(FRAC,GRAD)
      CALL NEWP(GRAD,X2)
      CALL BOUND(X1,X2,GRAD,DELTA)
*                                         ... IF DELTA IS SMALL, RETURN
         PLOW=PND1+DELTA
      IF(DELTA.GE.-DELMIN) THEN
         CALL OUTPUT (X1)
         RETURN
      END IF
*                                        ... IF DELTA IS LARGE, CONTINUE
      CALL PNDET(X2,FRAC,PND2)
      WRITE(11,'(/,3(2X,A5,F5.4))') 'PND1=',PND1,'PND2=',PND2,
     1                              'PLOW=',PLOW
*                                        ... LINE SEARCH FROM X1 TO X2, THE
*                                            MINIMIZING POINT IS X4
      CALL SEARCH(X1,PND1,X2,PND2,X4,FRAC,PND4)
*                                        ... UPDATE PND1 AND X1(J,T)
      PND1=PND4
      WRITE(11,'(1X,A6,F5.4)') 'PBEST=',PND1
      DO 17 T=1,TMAX-1
          DO 17 J=1,EP(NCELLS+1)-1
             X1(J,T)=X4(J,T)
   17 CONTINUE
      GO TO 15
      END


      SUBROUTINE MOVEF(TGTDN,TGTDNF)
**********************************************************************
*     PROGRAMMER: FRANK CALDWELL       DATE: SEP 87                  *
* PURPOSE:                                                           *
*   THIS SUBROUTINE CONDUCTS A MARKOV TRANSITION ONE PERIOD FORWARD IN *
* IN TIME. IT ESSENTIALLY CONDUCTS THE OPERATION OF POST-MULTIPLYING *
* THE ROW VECTOR OF TARGET PROBABILITY MASSES BY THE MARKOV TRANSITION *
* MATRIX.                                                            *
*                                                                    *
* INPUT:                                                             *
*   TGTDN: THE CURRENT TARGET DENSITY                                *
```

```
*                                                                      *
* OUTPUT:                                                              *
*   TGTDNF: THE FUTURE TARGET DENSITY AFTER ONE TRANSITION PERIOD      *
************************************************************************
*                                        ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH)
      REAL XO(NCELLS),TGSTRT(NCELLS),TRANS(LENGTH),A(LENGTH),
     1TGTDN(NCELLS),TGTDNF(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*
      DO 5 I=1,NCELLS
    5 TGTDNF(I)=0
      DO 10 I=1,NCELLS
         DO 10 J=EP(I),EP(I+1)-1
   10 TGTDNF(ADJ(J))=TGTDNF(ADJ(J))+TGTDN(I)*TRANS(J)
      RETURN
      END


      SUBROUTINE MOVEP(TGTDN,TGTDNP)
************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                     *
* PURPOSE:                                                             *
*   THIS SUBROUTINE CONDUCTS A MARKOV TRANSITION ONE PERIOD BACKWARD   *
* IN TIME. IT ESSENTIALLY CONDUCTS THE OPERATION OF POST-MULTIPLYING   *
* THE ROW VECTOR OF TARGET PROBABILITY MASSES BY THE MARKOV TRANSITION *
* MATRIX FOR TRANSITION BACKWARDS IN TIME.                             *
*                                                                      *
* INPUT:                                                               *
*   TGTDN: THE CURRENT TARGET DENSITY                                  *
*                                                                      *
* OUTPUT:                                                              *
*   TGTDNP: THE PAST TARGET DENSITY ONE TRANSITION PERIOD BACKWARDS IN *
*           IN TIME.                                                   *
************************************************************************
*                                        ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH)
      REAL XO(NCELLS),TGSTRT(NCELLS),TRANS(LENGTH),A(LENGTH),
     1TGTDN(NCELLS),TGTDNP(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*
      DO 5 I=1,NCELLS
    5 TGTDNP(I)=0
      DO 10 I=1,NCELLS
         DO 10 J=EP(I),EP(I+1)-1
   10 TGTDNP(I)=TGTDNP(I)+TGTDN(ADJ(J))*TRANS(J)
      RETURN
      END


      SUBROUTINE NEWP(GRAD,X2)
************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                     *
* PURPOSE:                                                             *
*   GIVEN THE VALUES OF GRAD(J,T), THIS SUBROUTINE LINEARIZES THE      *
* OBJECTIVE FUNCTION AND THEN FIND THE SHORTEST PATH THROUGH THE       *
* NETWORK VIA DYNAMIC PROGRAMMING. IT ALSO CALCULATES THE SET OF       *
* FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT SOLUTION, X2.       *
* THIS METHOD OF SOLUTION IS KNOWN AS THE FRANK-WOLFE PROCEDURE.       *
*                                                                      *
* INPUT:                                                               *
*   GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC     *
```

```
*         IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE    *
*         OBJECTIVE FUNCTION WITH RESPECT TO ARC J.                     *
*                                                                       *
* OUTPUT:                                                               *
*   x2: THE SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT.    *
*       THESE FLOWS ARE ALONG THE SHORTEST PATH THROUGH THE LINEARIZED *
*       NETWORK.                                                        *
*************************************************************************
*                                          ... DECLARE/INITALIZE
      INTEGER EPLEN,TMAX
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),NEXT(NCELLS,TMAX),T,ADD(LENGTH)
      REAL A(LENGTH),TRANS(LENGTH),VOC(NCELLS,TMAX),GRAD(LENGTH,TMAX),
     1XCELL(NCELLS),X2(LENGTH,TMAX),DUMMY(NCELLS),XO(NCELLS),
     2TGSTRT(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*                                          ... SET VOC(I,TMAX)=0
         DO 10 I=1,NCELLS
            VOC(I,TMAX)=0
            NEXT(I,TMAX)=0
   10    CONTINUE
*                                          ... CALCULATE THE VALUE OF
*                                              CONTINUING, VOC(I,T). KEEP
*                                              TRACK OF BEST DECISION WITH
*                                              ARRAY NEXT(I,T).
      DO 20 T=TMAX-1,1,-1
         DO 20 I=1,NCELLS
            VOC(I,T)=VOC(ADJ(EP(I)),T+1)+GRAD(EP(I),T)
            NEXT(I,T)=EP(I)
            DO 20 J=EP(I)+1,EP(I+1)-1
               TRIAL=VOC(ADJ(J),T+1)+GRAD(J,T)
               IF(TRIAL.LT.VOC(I,T)) THEN
                   VOC(I,T)=TRIAL
                   NEXT(I,T)=J
               END IF
   20 CONTINUE
****************** CALCULATE NEW FLOW, X2(J,T) ***********************
*                                         ... SET XCELL(I) EQUAL TO START(I)
*                                             WHERE XCELL(I) KEEPS TRACK OF
*                                             THE TOTAL SEARCH EFFORT IN
*                                             EACH CELL.
      DO 30 I=1,NCELLS
         DUMMY(I)=0
         XCELL(I)=XO(I)
   30 CONTINUE
      DO 35 T=1,TMAX-1
         DO 35 J=EP(1),EP(NCELLS+1)-1
            X2(J,T)=0
   35 CONTINUE
*                                          ... GENERATE X2(J,T) FROM
*                                              X2(J,T-1) AND NEXT(I,T).
      DO 50 T=1,TMAX-1
         DO 40 I=1,NCELLS
            J=NEXT(I,T)
            X2(J,T)=XCELL(I)
            DUMMY(ADJ(J))=DUMMY(ADJ(J))+X2(J,T)
  40     CONTINUE
*                                          ... RESET XCELL(I) FOR NEXT TIME
*                                              PERIOD.
         DO 50 I=1,NCELLS
            XCELL(I)=0
            XCELL(I)=DUMMY(I)
            DUMMY(I)=0
  50  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE PNDET(X,FRAC,PND)
************************************************************************
*      PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                    *
* PURPOSE:                                                             *
*   GIVEN A SET OF FLOWS, X, THIS PROGRAM CALCULATES THE PROBABILITY   *
* OF TARGET NONDETECTION.                                              *
*                                                                      *
* INPUT:                                                               *
*   X:  A SET OF FEASIBLE FLOWS                                        *
*                                                                      *
* OUTPUT:                                                              *
*   PND: THE PROBABILITY OF NONDETECTION                               *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH  *
*         TIME PERIOD.                                                 *
************************************************************************
*                                         ... INITIALIZE / DECLARE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),ADD(LENGTH),T
      REAL XO(NCELLS),TGSTRT(NCELLS),X(LENGTH,TMAX),FRAC(NCELLS,TMAX),
     1A(LENGTH),TRANS(LENGTH),TGTDN(NCELLS),TGTDNF(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*                                         ... DETERMINE THE FRACTION OF ALL
*                                             CELLS SEARCHED.
      CALL FRACT(X,FRAC)
*                                         ... ITERATIVELY CALCULATE THE
*                                             PROB. OF NONDETECTION PND.
*                                             SET THE TARGET DENSITY EQUAL
*                                             TO THE INITIAL DISTRIBUTION
      DO 10 I=1,NCELLS
   10   TGTDN(I)=TGSTRT(I)
       DO 20 T=1,TMAX
         DO 15 I=1,NCELLS
*                                         ... ACCOUNT FOR SEARCH
   15   TGTDN(I)=TGTDN(I)*EXP(-FRAC(I,T))
*                                         ... TRANSITION TO THE NEXT
*                                             TIME PERIOD
         CALL MOVEF(TGTDN,TGTDNF)
         DO 20 I=1,NCELLS
              TGTDN(I)=TGTDNF(I)
   20  CONTINUE
      PND=0
*                                         ... COMPUTE PND BY SUMMING ALL
*                                             REMAINING TARGET MASS
      DO 30 I=1,NCELLS
       PND=PND+TGTDN(I)
   30 CONTINUE
      RETURN
      END


      SUBROUTINE REACH(FRAC,R)
************************************************************************
*      PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                    *
* PURPOSE:                                                             *
*   THIS SUBROUTINE CALCULATES THE PROBABILITY OF REACHING CELL I IN   *
* TIME PERIOD T, R(I,T). NOTE THAT PROBABILITIES IN THE REACH MATRIX   *
* R(I,T) DO NOT ACCOUNT FOR THE SEARCH IN CELL I FOR TIME T.           *
*                                                                      *
* INPUT:                                                               *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH  *
*         TIME PERIOD.                                                 *
*                                                                      *
* OUTPUT:                                                              *
*  R: THE MATRIX OF DIMENSION NCELLS BY TMAX OF REACH PROBABILITIES    *
```

```
************************************************************************
*
*                                        ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH)
      REAL FRAC(NCELLS,TMAX),TGSTRT(NCELLS),R(NCELLS,TMAX),TRANS(LENGTH)
     1,A(LENGTH),XO(NCELLS),TGTDN(NCELLS),TGTDNF(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO, ADD
*                                        ... SET R(I,1) EQUAL TO THE
*                                            INITIAL TARGET DISTRIBUTION
      DO 5 I=1,NCELLS
         R(I,1)=TGSTRT(I)
    5 CONTINUE
*                                        ... COMPUTE R ITERATIVELY
      DO 20 T=1,TMAX-1
         DO 10 I=1,NCELLS
            TGTDN(I)= R(I,T)*EXP(-1*FRAC(I,T))
10       CONTINUE
         CALL MOVEF(TGTDN,TGTDNF)
         DO 20 I=1,NCELLS
               R(I,T+1)=TGTDNF(I)
20 CONTINUE
      RETURN
      END


      SUBROUTINE SEARCH(X1,PND1,X2,PND2,X4,FRAC,PND4)
************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                     *
* PURPOSE:                                                             *
*   THIS PROGRAM CONDUCTS A QUADRATIC LINE SEARCH ALONG THE LINE FROM  *
* START POINT X1 TO EXTREME POINT X2 FOR THE POINT THAT MINIMIZES THE  *
* PROBABILITY OF TARGET NONDETECTION. THIS MINIMIZING POINT IS THEN    *
* USED AS THE START POINT FOR THE NEXT FRANK-WOLFE ITERATION.          *
*                                                                      *
* INPUT:                                                               *
*   X1: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE START POINT        *
*   X2: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT      *
*   PND1: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*         BY X1.                                                       *
*   PND2: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*         BY X2.                                                       *
*                                                                      *
* OUTPUT:                                                              *
*   x4: THE SET OF FEASIBLE FLOWS ASSOCIATED WITH THE MINIMIZING POINT *
*       FROM THE QUADRATIC LINE SEARCH.                                *
*   FRAC: THE MATRIX SHOWING THE FRACTION OF CELL I SEARCHED DURING    *
*         TIME PERIOD T FOR THE SET OF FLOWS GIVEN BY X4.              *
*   PND4: THE PROBABILITY OF NONDETECTION FOR FLOWS GIVEN BY X4.       *
************************************************************************
*                                        ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH),COUNT
      REAL X1(LENGTH,TMAX),X2(LENGTH,TMAX),XO(NCELLS),X3(LENGTH,TMAX),
     1X4(LENGTH,TMAX),TGSTRT(NCELLS),TRANS(LENGTH),A(LENGTH),
     2FRAC(NCELLS,TMAX)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
      WRITE(11,*)'LINE SEARCH'
      COUNT=1
*                                        ... GENERATE X3 = .5*(X1+X2)
    5 DO 10 T=1,TMAX-1
               DO 10 J=1,EP(NCELLS+1)-1
                  X3(J,T)=.5*X1(J,T)+.5*X2(J,T)
   10 CONTINUE
      CALL PNDET(X3,FRAC,PND3)
```

```
*                                                ... GENERATE THETA
      THETA= .5*(-.75*PND1+PND3-.25*PND2)/(-.5*PND1+PND3-.5*PND2)
      IF(THETA.GE.1.0) THETA=1.0
      IF(THETA.LE..001) THETA=.001
*                                                ... GENERATE X4
      DO 20 T=1,TMAX-1
            DO 20 J=1,EP(NCELLS+1)-1
          X4(J,T)=THETA*X2(J,T)+(1-THETA)*X1(J,T)
   20 CONTINUE
      CALL PNDET(X4,FRAC,PND4)
      WRITE(11,'(3(2X,A5,F5.4),2X,A6,F5.4)') 'PND1=',PND1,'PND2=',PND2,
     1                                        'PND4=',PND4,'THETA=',THETA
*                                                ... CHECK TO NARROW INTERVAL
      IF((THETA.LT..1.OR.THETA.GT..9).AND.COUNT.LT.3) THEN
         COUNT=COUNT+1
         IF(THETA.LT..5) THEN
            IF(PND4.LE.PND3) THEN
                 PND2=PND3
                 DO 40 T=1,TMAX-1
                    DO 40 I=1,NCELLS
                       DO 40 J=EP(I),EP(I+1)-1
                          X2(J,T)=X3(J,T)
   40          CONTINUE
            ELSE
                 PND1=PND4
                 DO 50 T=1,TMAX-1
                    DO 50 I=1,NCELLS
                       DO 50 J=EP(I),EP(I+1)-1
                          X1(J,T)=X4(J,T)
   50          CONTINUE
            END IF
         ELSE
            IF(PND4.LE.PND3) THEN
                 PND1=PND3
                 DO 60 T=1,TMAX-1
                    DO 60 I=1,NCELLS
                       DO 60 J=EP(I),EP(I+1)-1
                          X1(J,T)=X3(J,T)
   60          CONTINUE
            ELSE
                 PND2=PND4
                 DO 70 T=1,TMAX-1
                    DO 70 I=1,NCELLS
                       DO 70 J=EP(I),EP(I+1)-1
                          X2(J,T)=X4(J,T)
   70          CONTINUE
            END IF
         END IF
         GO TO 5
      END IF
      RETURN
      END


      SUBROUTINE SURVIV(FRAC,S)
*************************************************************************
*     PROGRAMMER: FRANK CALDWELL       DATE: SEP 87                     *
* PURPOSE:                                                              *
*   THIS PROGRAM CALCULATES THE PROBABILITY OF SURVIVING TO TIME PERIOD*
* TMAX GIVEN THAT THE TARGET IS NONDETECTED IN CELL I BY TIME T.        *
*                                                                       *
* INPUT:                                                                *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH   *
*         TIME PERIOD.                                                  *
*                                                                       *
* OUTPUT:                                                               *
```

```
*   S: THE MATRIX OF DIMENSION NCELLS BY TMAX OF SURVIVE PROBABILITIES *
**********************************************************************
*
*                                         ... DECLARATIONS
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH)
      REAL FRAC(NCELLS,TMAX),S(NCELLS,TMAX),TRANS(LENGTH),A(LENGTH),
     1TGSTRT(NCELLS),XO(NCELLS),TGTDN(NCELLS),TGTDNP(NCELLS)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD
*                                         ... SET S(I,TMAX) = 1 FOR ALL I.
         DO 5 I=1,NCELLS
            S(I,TMAX)=1
    5    CONTINUE
*                                         ... ITERATIVELY CALCULATE S(I,T).
      DO 20 T=TMAX,2,-1
         DO 10 I=1,NCELLS
            TGTDN(I)=S(I,T)*EXP(-FRAC(I,T))
   10    CONTINUE
         CALL MOVEP(TGTDN,TGTDNP)
         DO 20 I=1,NCELLS
            S(I,T-1)=TGTDNP(I)
   20 CONTINUE
      RETURN
      END


      SUBROUTINE OUTPUT(X4)
**********************************************************************
*     PROGRAMMER: FRANK CALDWELL     DATE: SEP 87                      *
* PURPOSE:                                                             *
*   THIS PROGRAM PRINTS THE OPTIMAL SOLUTION FOR THE DIVISIBLE SEARCH  *
* EFFORT PROBLEM. OUTPUT IS PROVIDED IN THE FORM OF MATRICES DEPICT-   *
* ING THE GRID OF CELLS. THE FIRST SET OF MATRICES GIVES THE SEARCH    *
* EFFORT IN EACH CELL FOR ALL TIME PERIODS. THE SECOND SET GIVES THE   *
* REACH PROBABILITIES FOR EACH CELL FOR ALL TIME PERIODS. THESE TWO    *
* OUTPUTS ARE PROVIDED IN ORDER TO GIVE A REPRESENTATION OF THE TARGET *
* AND SEARCHER LOCATIONS THROUGHOUT THE PROBLEM.                       *
*                                                                      *
* INPUT:                                                               *
*   X: THE SET OF OPTIMAL FLOWS FOR THE PROBLEM                        *
*                                                                      *
* OUTPUT:                                                              *
*   OUTPUT IS TO A FILE, THERE ARE NO VARIABLES CALCULATED BY THIS     *
* PROGRAM FOR USE IN OTHER SUBROUTINES.                                *
**********************************************************************

*                                         ... DECLARE / INITIALIZE

      INTEGER EPLEN,TMAX
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=225)
      INTEGER ADJ(LENGTH),EP(EPLEN),ADD(LENGTH),T
      REAL A(LENGTH),TRANS(LENGTH),XO(NCELLS),TGSTRT(NCELLS),
     1XCELL(NCELLS),X4(LENGTH,TMAX),FRAC(NCELLS,TMAX),R(NCELLS,TMAX)
      COMMON EP,ADJ,TRANS,A,TGSTRT,XO,ADD

*                                         ... DETERMINE REACH AND FRAC
      CALL FRACT(X4,FRAC)
      CALL REACH(FRAC,R)
*                                         ... INITIALIZE XCELL(.), THIS
*                                             KEEPS TRACK OF THE FLOW
*                                             EFFORT IN EACH CELL
      DO 10 I=1,NCELLS
         XCELL(I)=XO(I)
   10 CONTINUE
      N=5
*                                         ... PRINT-OUT SEARCH EFFORT

      WRITE(11,'(A1,T10,A13)') '1','SEARCH EFFORT'
      T=1
```

```
      WRITE(11,100) 'TIME PERIOD',T
      DO 12 K=1,N
      WRITE(11,110) (XCELL(I), I=1+(K-1)*N,K*N)
   12 CONTINUE
      DO 30 T=1,TMAX-1
         DO 15 I=1,NCELLS
              XCELL(I)=0
   15    CONTINUE
         DO 20 I=1,NCELLS
              DO 20 J=EP(I),EP(I+1)-1
                   XCELL(ADJ(J))=XCELL(ADJ(J))+X4(J,T)
   20    CONTINUE
         WRITE(11,100) 'TIME PERIOD',T+1
      DO 22 K=1,N
      WRITE(11,110) (XCELL(I), I=1+(K-1)*N,K*N)
   22 CONTINUE
   30 CONTINUE
*                                      ... PRINT-OUT REACH PROBABILITIES
      WRITE(11,'(A1,T10,A19)') '1','REACH PROBABILITIES'
      DO 40 T=1,TMAX
         WRITE(11,100) 'TIME PERIOD',T
      DO 40 K=1,N
      WRITE(11,110) (R(I,T), I=1+(K-1)*N,K*N)
   40 CONTINUE
*
  100 FORMAT(T10,A11,1X,I2)
  110 FORMAT(15(2X,F5.3))
      RETURN
      END
```

APPENDIX C
BRANCH-AND-BOUND PROGRAM LISTING

1. SOME DETAILS ON PROGRAMMING METHODS

The branch-and-bound algorithm was coded in Fortran and run on the IBM 3033 computer just as the divisible search algorithm was. Because this procedure contains a modified version of the divisible program, all the comments written at the begining of Appendix B still apply. Within the branch-and-bound main program the trial paths are generated by the use of nested do loops the structure of which is very simple. Modifications to the divisible search effort program include:

- Updating the target's mass distribution for transitions and searches as specified by trial paths.
- Updating the time horizon to T − t periods for a trial path of length t.

Otherwise the divisile search program is essentially unchanged.

2. PROGRAM LISTING

```
      PROGRAM MAIN
*************************************************************************
*                             FRANK CALDWELL                            *
* PURPOSE:                                                              *
*    THIS IS THE CONTROLLING PROGRAM FOR THE CONSTRAINED SEARCH         *
* ALGORITHM. IT IMPLEMENTS THE BRANCHING PROCEDURE BY ESTABLISHING      *
* TRIAL INTEGER PATHS AND THEN CALLING THE SUBROUTINE LOWBND TO OBTAIN *
* A LOWER BOUND ON THE PROBABILITY OF NON-DETECTION FOR THAT TRIAL      *
* PATH.                                                                 *
*                                                                       *
* KEY VARIABLES:                                                        *
*      A: A MATRIX OF SEARCH EFFECTIVENESS PARAMETERS FOR EACH          *
*         ARC IN THE NETWORK. LISTED IN ADJACENCY LIST FORM.            *
*    ADD: A MATRIX SIMILAR TO THE ADJACENCY LIST, ADJ, BUT INSTEAD      *
*         OF GIVING THE HEADS OF ARCS INCIDENT TO CELLS LISTED IN THE   *
*         ENTRY POINT ARRAY, EP, THIS MATRIX GIVES THE TAILS OF ALL     *
*         ARCS THAT FLOW INTO THE CELL LISTED IN EP.                    *
*    ADJ: A MATRIX THAT GIVES THE HEADS OF ALL ARCS IN ADJACENCY LIST   *
*         FORMAT.                                                       *
*DELMIN: THE USER DEFINED INTERVAL OF ACCURACY REQUIRED FOR THE         *
*         LOWER BOUND. THIS ALSO SPECIFIES THE STOPPING CRITERIA        *
* DELTA: THE CHANGE IN PROBABILITY OF NONDETECTION PREDICTED BY THE     *
*         FIRST ORDER TAYLOR APPROXIMATION IN GOING FROM SOLUTION       *
*         X1 TO SOLUTION X2.                                            *
*     EP: THE ENTRY POINT ARRAY FOR THE ADJACENCY LISTS.                *
* EPLEN: A PARAMETER THAT IS USED TO SET THE DIMENSION OF THE EP ARRAY *
*   FRAC: A MATRIX OF DIMENSIONS NCELLS BY TMAX IN WHICH ENTRY,         *
*         FRAC(I,T) GIVES THE FRACTION OF CELL I SEARCHED IN TIME T.    *
*   GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC      *
*         IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE    *
*         OBJECTIVE FUNCTION WITH RESPECT TO ARC J FROM THE ADJ. LIST.  *
*     IT: THE DEPTH OF THE TRIAL PATH SPECIFIED AS A NUMBER OF PERIODS. *
* ITMAX: THE ADJUSTED TIME HORIZON FOR THE LOWBND SUBROUTINE.           *
*LENGTH: A PARAMETER USED TO SET DIMENSIONS OF ALL ADJACENY LISTS.      *
*NCELLS: A PARAMETER SPECIFYING THE NUMBER OF CELLS IN THE SQUARE GRID *
*   NEXT: AN ARRAY USED TO KEEP TRACK OF THE SHORTEST PATH.             *
*    PND: THE PROBABILITY OF NONDETECTION                               *
```

```
*  PND1: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN     *
*        BY X1 (THE START POINT).                                      *
*  PND2: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN     *
*        BY X2 (THE EXTREME POINT).                                    *
*  PND3: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN     *
*        BY X1 (THE MIDPOINT IN THE QUADRATIC LINE SEARCH).            *
*  PND4: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN     *
*        BY X4 (THE MINIMIZING POINT FROM THE LINE SEARCH).            *
*     R: THE MATRIX OF DIMENSION NCELLS BY TMAX OF REACH PROBS.        *
*     S: THE MATRIX OF DIMENSION NCELLS BY TMAX OF SURVIVE PROBS.      *
* START: THE SEARCHER'S INITIAL FEASIBLE SOLUTION                      *
*TGMASS: THE UPDATED TARGET MASS FOR TIME IT. ACCOUNTS FOR SEARCHES    *
*        AND TRANSITIONS UP TO TIME IT.                                *
*TGSTRT: A MATRIX GIVING THE TARGET STARTING DISTRIBUTION ON THE GRID. *
* TGTDN: THE CURRENT TARGET DENSITY.                                   *
*TGTDNF: THE FUTURE TARGET DENSITY AFTER ONE MARKOV TRANSITION.        *
*TGTDNP: THE PAST TARGET DENSITY ONE MARKOV TRANSITION BACKWARDS.      *
* THETA: THE FRACTION OF THE DISTANCE FROM X1 TO X2 THAT MINIMIZES     *
*        THE OBJECTIVE FUNCTION.                                       *
*  TMAX: THE TOTAL NUMBER OF SEARCH PERIODS.                           *
* TRANS: A MATRIX GIVING THE MARKOV TRANSITION PROBABILITIES FOR EACH  *
*        ARC LISTED IN ADJACENCY LIST FORMAT.                          *
* TRIAL: A DUMMMY VARIABLE USED TO KEEP TRACK OF VOC DURING THE SHORT- *
*        TEST PATH ROUTINE.                                            *
*   VOC: THE VALUE OF CONTINUING FOR EACH NODE ON THE SHORTEST PATH.   *
*     X: A SET OF ANY FEASIBLE FLOWS. (ALL FLOW VARIABLES ARE GIVEN    *
*        IN ADJACENCY LIST FORMAT.)                                    *
*    X1: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE START POINT.      *
*    X2: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT.    *
*    X3: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE MIDPOINT IN THE   *
*        QUADRATIC LINE SEARCH.                                        *
*    X4: THE SET OF FEASIBLE FLOWS ASSOCIATED WITH THE MINIMIZING POINT*
*        FROM THE QUADRATIC LINE SEARCH.                               *
*    XO: AN ARRAY GIVING THE THE AMOUNT OF SEARCH EFFORT IN EACH CELL. *
*                                                                      *
* REFERENCE:                                                           *
*    THE FORTRAN PROGRAM BB5x5 WRITTEN BY PROFESSOR JAMES EAGLE AT THE *
* NAVAL POSTGRADUATE SCHOOL IN MONTEREY, CALIFORNIA.                   *
************************************************************************

*                                         ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),IPATH(TMAX),ADD(LENGTH),BB(TMAX)
      REAL TRANS(LENGTH),A(LENGTH),TGSTRT(NCELLS),XO(NCELLS),
     1TGMASS(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*
*                                         ... INPUT DATA
      CALL INPUT(IPATH,TGSTRT,DELMIN)
*                                         ... CALCULATE INITIAL PBEST GIVEN
*                                             STARTING INTEGER PATH, IPATH
      DO 10 T=1,TMAX
         BB(T)=IPATH(T)
   10 CONTINUE
      CALL PNDETI(TGSTRT,TMAX+1,TGMASS)
      PBEST=0
      DO 20 I=1,NCELLS
         PBEST=PBEST+TGMASS(I)
   20 CONTINUE
      WRITE(06,'(1X,A14,1X,F5.4,10(2X,I2))') 'INITIAL PBEST=',PBEST,
     1                                       (BB(T), T=1,TMAX)
*                                         ... GENERATE A BEST SOLUTION
      IT=1
      CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
      DO 30 T=1,TMAX
         BB(T)=IPATH(T)
      CONTINUE
      WRITE(06,'(1X,A14,1X,F5.4,10(2X,I2))') 'INITIAL PBEST=', PBEST,
     1                                       (BB(T), T=1,TMAX)
```

```
*********************** BRANCH AND BOUND ******************************
*                                           ... INITIALIZE
      IT=1
      NTRIAL=0
      PLOW=0
      BB(1)=IPATH(1)
*                                           ... FORM TRIAL PATH BB(.) AND
*                                               FIND THE LOWER BOUND PLOW BY
*                                               CALLING LOWBND. IF PLOW IS
*                                               LESS THAN PBEST THE PATH IS
*                                               FATHOMED. OTHERWISE CONTINUE
*                                               DFS TO NEXT LEVEL.
      DO 200 J1=EP(BB(1)),EP(BB(1)+1)-1
         IT=2
         BB(IT)=ADJ(J1)
         NTRIAL=NTRIAL+1
         CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
         IF(PLOW.GE.PBEST) THEN
              GO TO 200
         END IF
*
      DO 201 J2=EP(BB(2)),EP(BB(2)+1)-1
         IT=3
         BB(IT)=ADJ(J2)
         NTRIAL=NTRIAL+1
         CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
         IF(PLOW.GE.PBEST) THEN
              GO TO 201
         END IF
*
      DO 202 J3=EP(BB(3)),EP(BB(3)+1)-1
         IT=4
         BB(IT)=ADJ(J3)
         NTRIAL=NTRIAL+1
         CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
         IF(PLOW.GE.PBEST) THEN
              GO TO 202
         END IF
*
      DO 203 J4=EP(BB(4)),EP(BB(4)+1)-1
         IT=5
         BB(IT)=ADJ(J4)
         NTRIAL=NTRIAL+1
         CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
         IF(PLOW.GE.PBEST) THEN
              GO TO 203
         END IF
*
      DO 204 J5=EP(BB(5)),EP(BB(5)+1)-1
         IT=6
         BB(IT)=ADJ(J5)
         NTRIAL=NTRIAL+1
         CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
         IF(PLOW.GE.PBEST) THEN
              GO TO 204
         END IF
*
      DO 205 J6=EP(BB(6)),EP(BB(6)+1)-1
         IT=7
         BB(IT)=ADJ(J6)
         NTRIAL=NTRIAL+1
         CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
         IF(PLOW.GE.PBEST) THEN
              GO TO 205
         END IF
*
      DO 206 J7=EP(BB(7)),EP(BB(7)+1)-1
         IT=8
```

```
            BB(IT)=ADJ(J7)
            NTRIAL=NTRIAL+1
            CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
            IF(PLOW.GE.PBEST) THEN
                GO TO 206
            END IF
*
        DO 207 J8=EP(BB(8)),EP(BB(8)+1)-1
            IT=9
            BB(IT)=ADJ(J8)
            NTRIAL=NTRIAL+1
            CALL LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
            IF(PLOW.GE.PBEST) THEN
                GO TO 207
            END IF
        DO 208 J9=EP(BB(9)),EP(BB(9)+1)-1
            IT=10
            BB(IT)=ADJ(J9)
            NTRIAL=NTRIAL+1
*                                             ... FOR TIME TMAX BB WILL SPECIFY
*                                                 A COMPLETE INTEGER PATH,
*                                                 THEREFORE CALL PNDETI INSTEAD
*                                                 OF LOWBND TO GET EXACT PND.
*
            CALL PNDETI(TGSTRT,TMAX+1,TGMASS)
            P=0
            DO 120 I=1,NCELLS
             P=P+TGMASS(I)
  120      CONTINUE
*                                             ... IF P IS LESS THAN PBEST BB IS
*                                                 A BETTER SOLUTION. UPDATE
*                                                 PBEST AND IPATH
            IF(P.LE.PBEST) THEN
               DO 100 T=1,TMAX
                 IPATH(T)=BB(T)
  100         CONTINUE
               WRITE(05,'(A6,F5.4,10(2X,I2))')'PBEST=',PBEST,
     1             (IPATH(T), T=1,TMAX)
               PBEST=P
            END IF
*
  208 CONTINUE
  207 CONTINUE
  206 CONTINUE
  205 CONTINUE
  204 CONTINUE
  203 CONTINUE
  202 CONTINUE
  201 CONTINUE
  200 CONTINUE
*                                             ... OUTPUT RESULTS
        WRITE(06,'(1X,A20,1X,I4)') 'TOTAL TRIAL PATHS = ',NTRIAL
        STOP
        END


        SUBROUTINE BOUND(X1,X2,GRAD,DELTA)
*******************************************************************************
*       PROGRAMMER: FRANK CALDWELL       DATE: SEP 87                         *
* PURPOSE:                                                                    *
*   THIS PROGRAM COMPUTES THE DELTA FOR USE IN CALCULATING THE LOWER          *
* BOUND ASSOCIATED WITH EACH FRANK-WOLFE ITERATION. THIS DELTA IS THE         *
* CHANGE IN THE PROBABILITY OF NONDETECTION ACHIEVED BY GOING FROM X1         *
* TO X2 AND IS CALCULATED BY THE FIRST ORDER TAYLOR APPROXIMATION.            *
*                                                                             *
* INPUT:                                                                      *
*   X1: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE START POINT               *
*   X2: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT             *
```

```
*   GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC       *
*         IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE     *
*         OBJECTIVE FUNCTION WITH RESPECT TO ARC J.                      *
* OUTPUT:                                                                *
*   DELTA: THE CHANGE IN PROBABILITY OF NONDETECTION PREDICTED BY THE    *
*         FIRST ORDER TAYLOR APPROXIMATION IN GOING FROM SOLUTION        *
*         X1 TO SOLUTION X2.                                             *
**************************************************************************
*                                         ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH),BB(TMAX)
      REAL X1(LENGTH,TMAX),X2(LENGTH,TMAX),GRAD(LENGTH,TMAX),XO(NCELLS),
     1TRANS(LENGTH),A(LENGTH)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
      DELTA=0.0
      DO 10 T=1,ITMAX-1
            DO 10 J=1,EP(NCELLS+1)-1
              DELTA=DELTA+GRAD(J,T)*(X2(J,T)-X1(J,T))
   10 CONTINUE
      RETURN
      END


      SUBROUTINE FRACT(X,FRAC)
**************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                       *
* PURPOSE:                                                               *
*   THIS PROGRAM CALCULATES THE FRACTION OF CELL I SEARCHED IN TIME      *
* PERIOD T. THIS FRACTION IS SIMPLY THE SUM OF OVER ALL ADJACENT CELLS   *
* OF THE PRODUCT OF FLOW EFFORT AND SEARCH EFFECTIVENESS.                *
*                                                                        *
* INPUT:                                                                 *
*   X:  A SET OF FEASIBLE FLOWS                                          *
*                                                                        *
* OUTPUT:                                                                *
*  FRAC: A MATRIX OF DIMENSIONS NCELLS BY TMAX IN WHICH ENTRY,           *
* FRAC(I,T) GIVES THE FRACTION OF CELL I SEARCHED IN TIME PERIOD T.      *
**************************************************************************
*
*                                         ... DECLARATIONS
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH),BB(TMAX)
      REAL A(LENGTH),FRAC(NCELLS,TMAX),TRANS(LENGTH),X(LENGTH,TMAX),
     1XO(NCELLS),TGMASS(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*                                         ... COMPUTE FRACTION OF CELL I
*                                             SEARCHED IN TIME PERIOD T
      DO 10 I=1,NCELLS
*                                         ... ASSUME SEARCHER'S STARTING
*                                             CELL IS COMPLETELY SEARCHED
                     FRAC(I,1)=XO(I)
         DO 10 T=2,ITMAX
*                                         ... FOR OTHER TIME PERIODS SUM
*                                             TOTAL FLOW INTO CELL I
               FRAC(I,T)=0
               DO 10 J=EP(I),EP(I+1)-1
                     FRAC(I,T)=FRAC(I,T)+A(ADD(J))*X(ADD(J),T-1)
   10          CONTINUE
      RETURN
      END


      SUBROUTINE GRADF(TGMASS,FRAC,GRAD)
**************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                       *
```

```
* PURPOSE:                                                                  *
*   THIS PROGRAM CALCULATES THE PARTIAL DERIVATIVES OF THE OBJECTIVE       *
* FUCTION WITH RESPECT TO EACH ARC IN THE NETWORK.                         *
*                                                                          *
* INPUT:                                                                   *
*   TGMASS: THE UPDATED TARGET DISTRIBUTION FOR TIME IT. THIS ACCOUNTS     *
*           FOR ALL SEARCHES AND TRANSITIONS UP TO AND INCLUDING IT.       *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH      *
*         TIME PERIOD.                                                     *
*                                                                          *
* OUTPUT:                                                                  *
*   GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC         *
*         IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE       *
*         OBJECTIVE FUNCTION WITH RESPECT TO ARC J.                        *
****************************************************************************
*
*                                          ... DECLARE / INITIALIZE
*
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH),BB(TMAX)
      REAL A(LENGTH),R(NCELLS,TMAX),S(NCELLS,TMAX),FRAC(NCELLS,TMAX),
     1GRAD(LENGTH,TMAX),TRANS(LENGTH),X(LENGTH,TMAX),XO(NCELLS),
     2TGMASS(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*                                          ... CALL REACH,SURVIV
      CALL REACH(TGMASS,FRAC,R)
      CALL SURVIV(FRAC,S)
*                                          ... CALCULATE PARTIAL DERIVATIVES
*                                              FOR EACH FLOW X(.,T)
      DO 10 T=1,ITMAX-1
         DO 10 I=1,NCELLS
            DUMMY=-R(I,T+1)*EXP(-1*FRAC(I,T+1))*S(I,T+1)
            DO 10 J=EP(I),EP(I+1)-1
               GRAD(ADD(J),T)=DUMMY*A(ADD(J))
   10 CONTINUE
      RETURN
      END


      SUBROUTINE INPUT(IPATH,TGSTRT,DELMIN)
****************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                         *
* PURPOSE:                                                                 *
*   THIS PROGRAM READS IN DATA FROM AN INPUT FILE FOR USE FOR THE          *
* CONSTRAINED SEARCH ALGORITHM.                                            *
****************************************************************************
*                                          ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER NADJ,EP(EPLEN),ADJ(LENGTH),IPATH(TMAX),T,ADD(LENGTH),
     1BB(TMAX)
      REAL TRANS(LENGTH),A(LENGTH),TGSTRT(NCELLS),XO(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*                                          ... READ IN DELMIN, THE DESIRED
*                                              ACCURACY OF THE LOWER BOUND
      READ(01,*) DELMIN
*
*                                          ... READ IN ADJACENT CELL NUMBERS
*                                              IN ADJACENCY LIST FORM WITH
*                                              ENTRY POINT ARRAY, EP(.), AND
*                                              HEADARRAY, ADJ(.)
      T=1
      DO 5 I=1,NCELLS
         EP(I)=T
         READ(01,*) DUMMY,NADJ,(ADJ(J), J=T,T+NADJ-1)
         T=T+NADJ
```

```
    5 CONTINUE
      EP(NCELLS+1)= T
      ADJ(T)=0
*                                         ... FOR EACH CELL I GENERATE THE
*                                             ADDRESS ARRAY ADD(.) WHICH
*                                             GIVES THE ADJACENCY LIST
*                                             POSITIONS FOR ALL FLOWS INTO
*                                             CELL I
      L=1
      DO 8 I=1,NCELLS
         DO 8 K=1,LENGTH
              IF(ADJ(K).EQ.I) THEN
                   ADD(L)=K
                   L=L+1
              END IF
    8 CONTINUE
      ADD(L)=0
*                                         ... READ IN TARGET TRANSITION
*                                             PROBABILITIES TRANS (.) IN
*                                             ADJACENCY LIST FORM
      DO 10 I=1,NCELLS
         READ(01,*) DUMMY,(TRANS(J), J=EP(I),EP(I+1)-1)
   10 CONTINUE
      TRANS(EP(NCELLS+1))=0
*                                         ... READ IN SEARCH EFFECTIVENESS
*                                             A(.), IN ADJACENCY LIST FORM
      DO 20 I=1,NCELLS
         READ(01,*) DUMMY,(A(J), J=EP(I),EP(I+1)-1)
   20 CONTINUE
      A(EP(NCELLS+1))=0
*                                         ... READ IN STARTING SOLUTION,
*                                             IPATH(.)
*
      READ(01,*) (IPATH(T), T=1,TMAX)
*
*                                         ... READ IN INITIAL TARGET
*                                             DISTRIBUTION, TGSTRT(.)
      DO 30 I=1,NCELLS
   30   READ(01,*) DUMMY,TGSTRT(I)
*
   11 RETURN
      END


         SUBROUTINE LOWBND(DELMIN,TGSTRT,IT,PLOW,PBEST,IPATH)
***************************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                        *
* PURPOSE:                                                                *
*   THIS SUBROUTINE REPRESENTS THE SUBSTANTIAL PART OF THE DIVISIBLE      *
* EFFORT PROGRAM. IT CONTROLS THE ITERATIVE SEQUENCE OF THE SOLUTION      *
* TECHNIQUE BY CALLING VARIOUS SUBROUTINES TO LINEARIZE THE OBJECTIVE     *
* FUNCTION, FIND THE SHORTEST PATH, ETC.                                  *
*                                                                         *
* INPUTS:                                                                 *
*   DELMIN: THE USER DEFINED INTERVAL OF ACCURACY REQUIRED FOR THE        *
*           LOWER BOUND. THIS ALSO SPECIFIES THE STOPPING CRITERIA        *
*   TGSTRT: A MATRIX GIVING THE TARGET STARTING DISTRIBUTION              *
*           ON THE GRID.                                                  *
*       IT: AN INTEGER GIVING THE DEPTH OF THE TRIAL PATH.                *
*    PBEST: THE CURRENT BEST PROBABILITY OF NONDETECTION.                 *
*    IPATH: THE CURRENT BEST INTEGER PATH CORRESPONDING TO PBEST.         *
*                                                                         *
* OUTPUTS:                                                                *
*     PLOW: THE VALUE OF THE LOWER BOUND.                                 *
***************************************************************************
```

```
*
*                                              ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER MARK
      INTEGER  EP(EPLEN),ADJ(LENGTH),BB(TMAX),T,ADD(LENGTH),IPATH(TMAX)
      REAL X1(LENGTH,TMAX),XO(NCELLS),FRAC(NCELLS,TMAX),
     1TGMASS(NCELLS),S(NCELLS,TMAX),A(LENGTH),GRAD(LENGTH,TMAX),
     2TRANS(LENGTH),X2(LENGTH,TMAX),X4(LENGTH,TMAX),TGSTRT(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*                                              ... RESET TIME HORIZON
      ITMAX=TMAX-IT+1
      ICOUNT=1
      CALL PNDETI(TGSTRT,IT,TGMASS)
*
*                                              ... CALCULATE FLOW X1 FOR THE
*                                                  INITIAL FEASIBLE SOLUTION.
*                                                  XO(I) IS THE STARTING SEARCH
*                                                  EFFORT IN CELL I
      DO 10 I=1,NCELLS
           XO(I)=0
   10 CONTINUE
      XO(BB(IT))=1
      INDEX=EP(BB(IT))
      DO 12 T=1,ITMAX-1
         DO 11 J=1,EP(NCELLS+1)-1
              X1(J,T)=0
   11    CONTINUE
         X1(INDEX,T)=1
         INDEX=EP(ADJ(INDEX))
   12 CONTINUE
*
************ LOWER BOUND AND FRANK-WOLFE (F.W.) METHOD ****************
*                                              ... FIND PND1, INITIAL NON-
*                                                  DETECTION PROBABILITY
      CALL PNDET(TGMASS,X1,FRAC,PND1)
*                                              ... IF PND1 IS LESS THAN PBEST
*                                                  THE TRIAL PATH CANNOT BE
*                                                  FATHOMED; RETURN.
   15 IF((PND1.LT.PBEST).AND.(IT.NE.1)) GO TO 20
*                                              ... IF NOT...CONTINUE WITH FRANK-
*                                                  WOLFE METHOD. LINEARIZE OBJ.
*                                                  FUNCTION AND FIND THE EXTREME
*                                                  POINT SOLUTION, X2.
      CALL GRADF(TGMASS,FRAC,GRAD)
      CALL NEWP(TGMASS,GRAD,X2)
*                                              ... COMPUTE THE LOWER BOUND PLOW
*                                                  FOR THE CURRENT SOLUTION
      CALL BOUND(X1,X2,GRAD,DELTA)
         PLOW=PND1+DELTA
*                                              ... IF PLOW IS GREATER THEN PBEST
*                                                  THE TRIAL PATH IS FATHOMED.
*                                                  ALSO, IF PLOW IS KNOWN WITHIN
*                                                  DELMIN RETURN.
      IF((DELTA.GE.-DELMIN).OR.(PLOW.GT.PBEST)) GOTO 20
*                                              ... IF NOT...CHECK EXTREME POINT
*                                                  SOLUTION X2 TO SEE IF IT IS
*                                                  BETTER THAN CURRENT PBEST. BY
*                                                  F.W. THIS IS GUARANTEED TO BE
*                                                  AN INTEGER SOLUTION.
      CALL PNDET(TGMASS,X2,FRAC,PND2)
      IF(PND2.LT.PBEST) THEN
         PBEST=PND2
         DO 16 T=1,IT
             IPATH(T)=BB(T)
   16    CONTINUE
         DO 17 T=2,ITMAX
```

```
            DO 17 I=1,NCELLS
               IF(FRAC(I,T).GT..97) THEN
                  IPATH(IT+T-1)=I
               END IF
17          CONTINUE
            IF (IT.NE.1) WRITE(05,'(A6,F5.4,10(2X,I2))')'PBEST=',PBEST,
     1                    (IPATH(T), T=1,TMAX)
      END IF
*                                          ... LINE SEARCH FROM X1 TO X2
*                                              RESULT IS X4 AND PND4
      CALL SEARCH(TGMASS,X1,PND1,X2,PND2,X4,FRAC,PND4)
*
*                                          ... IF IMPROVEMENT FROM PND1 TO
*                                              PND4 IS SMALL THEN RETURN.
*                                              THIS STOPS F.W. IN THE TAILS.
      IF (ICOUNT.GT.50) GO TO 20
*                                          ... UPDATE PND1 AND X1(J,T)
*                                              AND CONTINUE WITH F.W.
      PND1=PND4
      DO 27 T=1,ITMAX-1
          DO 27 J=1,EP(NCELLS+1)-1
             X1(J,T)=X4(J,T)
27    CONTINUE
      ICOUNT=ICOUNT+1
      GO TO 15
20    IF(IT.NE.1) WRITE(06,100) PBEST,PND1,PLOW,ICOUNT, (BB(T), T=1,IT)
100   FORMAT(3(2X,F5.4),2X,I2,3X,10(I2,1X))
      RETURN
      END


      SUBROUTINE MOVEF(TGTDN,TGTDNF)
***********************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                    *
* PURPOSE:                                                            *
*   THIS SUBROUTINE CONDUCTS A MARKOV TRANSITION ONE PERIOD FORWARD IN *
* IN TIME. IT ESSENTIALLY CONDUCTS THE OPERATION OF POST-MULTIPLYING  *
* THE ROW VECTOR OF TARGET PROBABILITY MASSES BY THE MARKOV TRANSITION *
* MATRIX.                                                             *
*                                                                     *
* INPUT:                                                              *
*   TGTDN: THE CURRENT TARGET DENSITY                                 *
*                                                                     *
* OUTPUT:                                                             *
*   TGTDNF: THE FUTURE TARGET DENSITY AFTER ONE TRANSITION PERIOD     *
***********************************************************************
*                                          ... DECLARE / INITIALIZE
      INTEGER EPLEN,TMAX
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),BB(TMAX),ADD(LENGTH)
      REAL XO(NCELLS),TRANS(LENGTH),A(LENGTH),TGTDN(NCELLS),
     1TGTDNF(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*
      DO 5 I=1,NCELLS
5      TGTDNF(I)=0
      DO 10 I=1,NCELLS
         DO 10 J=EP(I),EP(I+1)-1
            TGTDNF(ADJ(J))=TGTDNF(ADJ(J))+TGTDN(I)*TRANS(J)
10       CONTINUE
      RETURN
      END


      SUBROUTINE MOVEP(TGTDN,TGTDNP)
***********************************************************************
```

```
*       PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                       *
* PURPOSE:                                                                 *
*   THIS SUBROUTINE CONDUCTS A MARKOV TRANSITION ONE PERIOD BACKWARD       *
* IN TIME. IT ESSENTIALLY CONDUCTS THE OPERATION OF POST-MULTIPLYING       *
* THE ROW VECTOR OF TARGET PROBABILITY MASSES BY THE MARKOV TRANSITION     *
* MATRIX FOR TRANSITION BACKWARDS IN TIME.                                 *
*                                                                          *
* INPUT:                                                                   *
*   TGTDN: THE CURRENT TARGET DENSITY                                      *
*                                                                          *
* OUTPUT:                                                                  *
*   TGTDNP: THE PAST TARGET DENSITY ONE TRANSITION PERIOD BACKWARDS IN     *
*           IN TIME.                                                       *
****************************************************************************
*                                               ... DECLARE / INITIALIZE
      INTEGER EPLEN,TMAX
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),BB(TMAX),ADD(LENGTH)
      REAL XO(NCELLS),TRANS(LENGTH),A(LENGTH),TGTDN(NCELLS),
     1TGTDNP(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*
      DO 5 I=1,NCELLS
    5  TGTDNP(I)=0
      DO 10 I=1,NCELLS
          DO 10 J=EP(I),EP(I+1)-1
             TGTDNP(I)=TGTDNP(I)+TGTDN(ADJ(J))*TRANS(J)
   10     CONTINUE
      RETURN
      END


      SUBROUTINE NEWP(TGMASS,GRAD,X2)
****************************************************************************
*       PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                       *
* PURPOSE:                                                                 *
*   GIVEN THE VALUES OF GRAD(J,T), THIS SUBROUTINE LINEARIZES THE          *
* OBJECTIVE FUNCTION AND THEN FIND THE SHORTEST PATH THROUGH THE           *
* NETWORK VIA DYNAMIC PROGRAMMING. IT ALSO CALCULATES THE SET OF           *
* FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT SOLUTION, X2.           *
* THIS METHOD OF SOLUTION IS KNOWN AS THE FRANK-WOLFE PROCEDURE.           *
*                                                                          *
* INPUT:                                                                   *
*   TGMASS: THE UPDATED TARGET DISTRIBUTION FOR TIME IT. THIS ACCOUNTS     *
*           FOR ALL SEARCHES AND TRANSITIONS UP TO AND INCLUDING IT.       *
*   GRAD: A MATRIX OF PARTIAL DERIVATIVES WITH RESPECT TO EACH ARC         *
*         IN THE NETWORK. GRAD(J,T) IS THE PARTIAL DERIVATIVE OF THE       *
*         OBJECTIVE FUNCTION WITH RESPECT TO ARC J.                        *
*                                                                          *
* OUTPUT:                                                                  *
*   x2: THE SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT.       *
*       THESE FLOWS ARE ALONG THE SHORTEST PATH THROUGH THE LINEARIZED     *
*       NETWORK.                                                           *
****************************************************************************
*                                               ... DECLARE/INITALIZE
      INTEGER EPLEN,TMAX
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),NEXT(NCELLS,TMAX),T,ADD(LENGTH),
     1BB(TMAX)
      REAL A(LENGTH),TRANS(LENGTH),VOC(NCELLS,TMAX),GRAD(LENGTH,TMAX),
     1XCELL(NCELLS),X2(LENGTH,TMAX),DUMMY(NCELLS),XO(NCELLS),
     2TGMASS(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
********************* FIND SHORTEST PATH *****************************
*                                               ... SET VOC(I,TMAX)=0
          DO 10 I=1,NCELLS
```

```
              VOC(I,ITMAX)=0
              NEXT(I,ITMAX)=0
   10     CONTINUE
*                                                 ... CALCULATE THE VALUE OF
*                                                     CONTINUING, VOC(I,T). KEEP
*                                                     TRACK OF BEST DECISION WITH
*                                                     ARRAY NEXT(I,T).
      DO 20 T=ITMAX-1,1,-1
          DO 20 I=1,NCELLS
              VOC(I,T)=VOC(ADJ(EP(I)),T+1)+GRAD(EP(I),T)
              NEXT(I,T)=EP(I)
              DO 20 J=EP(I)+1,EP(I+1)-1
                  TRIAL=VOC(ADJ(J),T+1)+GRAD(J,T)
                  IF(TRIAL.LT.VOC(I,T)) THEN
                       VOC(I,T)=TRIAL
                       NEXT(I,T)=J
                  END IF
   20 CONTINUE
****************** CALCULATE NEW FLOW, X2(J,T) ***********************
*                                                ... SET XCELL(I) EQUAL TO START(I)
*                                                    WHERE XCELL(I) KEEPS TRACK OF
*                                                    THE TOTAL SEARCH EFFORT IN
*                                                    EACH CELL.
      DO 30 I=1,NCELLS
          XCELL(I)=XO(I)
          DUMMY(I)=0
   30 CONTINUE
      DO 35 T=1,ITMAX-1
          DO 35 J=EP(1),EP(NCELLS+1)-1
                X2(J,T)=0
   35 CONTINUE
*                                                 ... GENERATE X2(J,T) FROM
*                                                     X2(J,T-1) AND NEXT(I,T).
      DO 50 T=1,ITMAX-1
          DO 40 I=1,NCELLS
                       J=NEXT(I,T)
                       X2(J,T)=XCELL(I)
                       DUMMY(ADJ(J))=DUMMY(ADJ(J))+X2(J,T)
  40      CONTINUE
*             PRINT *, 'XCELL',XCELL
*                                                 ... RESET XCELL(I) FOR NEXT TIME
*                                                     PERIOD.
          DO 50 I=1,NCELLS
              XCELL(I)=0
              XCELL(I)=DUMMY(I)
              DUMMY(I)=0
  50  CONTINUE
      RETURN
      END


      SUBROUTINE PNDET(TGMASS,X,FRAC,PND)
***********************************************************************
*     PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                    *
* PURPOSE:                                                            *
*   GIVEN A SET OF FLOWS, X, THIS PROGRAM CALCULATES THE PROBABILITY  *
* OF TARGET NONDETECTION.                                             *
*                                                                     *
* INPUT:                                                              *
*   TGMASS: THE UPDATED TARGET DISTRIBUTION FOR TIME IT. THIS ACCOUNTS *
*          FOR ALL SEARCHES AND TRANSITIONS UP TO AND INCLUDING IT.   *
*   X:  A SET OF FEASIBLE FLOWS                                       *
*                                                                     *
* OUTPUT:                                                             *
*   PND: THE PROBABILITY OF NONDETECTION                              *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH *
*         TIME PERIOD.                                                *
***********************************************************************
```

```
*                                                  ... INITIALIZE / DECLARE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),ADD(LENGTH),BB(TMAX),T
      REAL XO(NCELLS),TGMASS(NCELLS),X(LENGTH,TMAX),FRAC(NCELLS,TMAX),
     1A(LENGTH),TRANS(LENGTH),TGTDN(NCELLS),TGTDNF(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*                                                  ... DETERMINE THE FRACTION OF
*                                                      EACH CELL THAT IS SEARCHED
      CALL FRACT(X,FRAC)
*                                                  ... INITIAL TARGET DENSITY IS SET
*                                                      EQUAL TO THE STARTING TARGET
*                                                      MASS
      DO 10 I=1,NCELLS
         TGTDN(I)=TGMASS(I)
   10 CONTINUE
      DO 20 T=1,ITMAX
         DO 15 I=1,NCELLS
*                                                  ... ACCOUNT FOR SEARCH IN TIME
*                                                      PERIOD T
                TGTDN(I)=TGTDN(I)*EXP(-FRAC(I,T))
   15    CONTINUE
*                                                  ... MOVE TARGET DENSITY FORWARD
*                                                      IN TIME ACCORDING TO MARKOV
*                                                      TRANSITION MATRIX
         CALL MOVEF(TGTDN,TGTDNF)
*                                                  ... UPDATE TARGET DENSITY
         DO 20 I=1,NCELLS
                TGTDN(I)=TGTDNF(I)
   20 CONTINUE
*                                                  ... AFTER ITMAX TIME PERIODS OF
*                                                      SEARCH, SUM REMAINING TARGET
*                                                      MASS TO FIND PND
      PND=0
      DO 30 I=1,NCELLS
       PND=PND+TGTDN(I)
   30 CONTINUE
      RETURN
      END


      SUBROUTINE PNDETI(TGSTRT,IT,TGMASS)
***********************************************************************
*      PROGRAMMER: FRANK CALDWELL       DATE: SEP 87                   *
* PURPOSE:                                                             *
*       GIVEN AN INTEGER SOLUTION THIS PROGRAM FINDS THE DISTRIBUtTION *
* OF THE REMAINING TARGET MASS FOR TIME PERIOD IT ACCOUNTING FOR  ALL  *
* SEARCHES AND TRANSITIONS UP TO THE START OF TIME PERIOD IT.          *
*                                                                      *
* INPUTS:                                                              *
*  TGSTRT: A MATRIX GIVING THE TARGET STARTING DISTRIBUTION ON THE     *
*          GRID.                                                       *
*      IT: THE DEPTH OF THE TRIAL PATH.                                *
*                                                                      *
* OUTPUTS:                                                             *
*  TGMASS: THE UPDATED TARGET MASS FOR TIME IT.                        *
*                                                                      *
***********************************************************************
*                                                  ... DECLARE / INITIALIZE
      INTEGER EPLEN,TMAX
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,BB(TMAX),ADD(LENGTH)
      REAL XO(NCELLS),TGMASS(NCELLS),TRANS(LENGTH),A(LENGTH),
     1FRACI(NCELLS),FMASS(NCELLS),TGSTRT(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
```

```
      DC 10 I=1,NCELLS
         TGMASS(I)=TGSTRT(I)
   10 CONTINUE
      IF(IT.EQ.1) GO TO 50
      DO 30 T=1,IT-1
*                                         ... ACCOUNT FOR SEARCH
         TGMASS(BB(T))=TGMASS(BB(T))*EXP(-1.0)
*                                         ... TRANSITION FORWARD IN TIME
         CALL MOVEF(TGMASS,FMASS)
*                                         ... UPDATE
         DO 30 I=1,NCELLS
               TGMASS(I)=FMASS(I)
   30 CONTINUE
   50 RETURN
      END


      SUBROUTINE REACH(TGMASS,FRAC,R)
*************************************************************************
*     PROGRAMMER: FRANK CALDWELL     DATE: SEP 87                       *
* PURPOSE:                                                              *
*   THIS SUBROUTINE CALCULATES THE PROBABILITY OF REACHING CELL I IN    *
* TIME PERIOD T. R(I,T). NOTE THAT PROBABILITIES IN THE REACH MATRIX    *
* R(I,T) DO NOT ACCOUNT FOR THE SEARCH IN CELL I FOR TIME T.            *
*                                                                       *
* INPUT:                                                                *
*   TGMASS: THE UPDATED TARGET DISTRIBUTION FOR TIME IT. THIS ACCOUNTS *
*           FOR ALL SEARCHES AND TRANSITIONS UP TO AND INCLUDING IT.    *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH  *
*         TIME PERIOD.                                                  *
*                                                                       *
* OUTPUT:                                                               *
*  R: THE MATRIX OF DIMENSION NCELLS BY TMAX OF REACH PROBABILITIES    *
*************************************************************************
*
*                                         ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH),BB(TMAX)
      REAL FRAC(NCELLS,TMAX),TGMASS(NCELLS),R(NCELLS,TMAX),TRANS(LENGTH)
     1,A(LENGTH),XO(NCELLS),TGTDN(NCELLS),TGTDNF(NCELLS)
      COMMON EP,ADJ,TRANS,A,XC, ADD,ITMAX,BB
*                                          ... FOR TIME PERIOD 1 SET R(I,1)
*                                              EQUAL TO THE PROBABILITY
*                                              THE TARGET STARTS IN CELL I
      DO 5 I=1,NCELLS
         R(I,1)=TGMASS(I)
    5 CONTINUE
*                                         ... ITERATIVELY CALCULATE R(I,T)
*                                             FOR ALL OTHER TIME PERIODS
      DO 20 T=1,ITMAX-1
*                                         ... ACCOUNT FOR SEARCH IN TIME T
         DO 10 I=1,NCELLS
               TGTDN(I)=R(I,T)*EXP(-1.0*FRAC(I,T))
   10    CONTINUE
*                                         ... MOVE DENSITY FORWARD IN TIME
         CALL MOVEF(TGTDN,TGTDNF)
*                                         ... SET R(I,T+1)
         DO 20 I=1,NCELLS
               R(I,T+1)=TGTDNF(I)
   20 CONTINUE
      RETURN
      END

      SUBROUTINE SEARCH(TGMASS,X1,PND1,X2,PND2,X4,FRAC,PND4)
*************************************************************************
```

```
*      PROGRAMMER: FRANK CALDWELL      DATE: SEP 87                    *
* PURPOSE:                                                             *
*   THIS PROGRAM CONDUCTS A QUADRATIC LINE SEARCH ALONG THE LINE FROM  *
* START POINT X1 TO EXTREME POINT X2 FOR THE POINT THAT MINIMIZES THE  *
* PROBABILITY OF TARGET NONDETECTION. THIS MINIMIZING POINT IS THEN    *
* USED AS THE START POINT FOR THE NEXT FRANK-WOLFE ITERATION.          *
*                                                                      *
* INPUT:                                                               *
*   TGMASS: THE UPDATED TARGET DISTRIBUTION FOR TIME IT. THIS ACCOUNTS *
*          FOR ALL SEARCHES AND TRANSITIONS UP TO AND INCLUDING IT.    *
*   X1: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE START POINT        *
*   X2: A SET OF FEASIBLE FLOWS ASSOCIATED WITH THE EXTREME POINT      *
*   PND1: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*         BY X1.                                                       *
*   PND2: THE PROBABILITY OF NONDETECTION FOR SEARCH FLOWS AS GIVEN    *
*         BY X2.                                                       *
*                                                                      *
* OUTPUT:                                                              *
*   x4: THE SET OF FEASIBLE FLOWS ASSOCIATED WITH THE MINIMIZING POINT *
*       FROM THE QUADRATIC LINE SEARCH.                                *
*   FRAC: THE MATRIX SHOWING THE FRACTION OF CELL I SEARCHED DURING    *
*        TIME PERIOD T FOR THE SET OF FLOWS GIVEN BY X4.               *
*   PND4: THE PROBABILITY OF NONDETECTION FOR FLOWS GIVEN BY X4.       *
************************************************************************
*                                           ... DECLARE / INITIALIZE
      INTEGER TMAX,EPLEN
      PARAMETER (NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH),COUNT,BB(TMAX)
      REAL X1(LENGTH,TMAX),X2(LENGTH,TMAX),XO(NCELLS),X3(LENGTH,TMAX),
     1X4(LENGTH,TMAX),TGMASS(NCELLS),TRANS(LENGTH),A(LENGTH),
     2FRAC(NCELLS,TMAX)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
      COUNT=1
*                                           ... GENERATE X3 = .5*(X1+X2)
    5 DO 10 T=1,ITMAX-1
              DO 10 J=1,EP(NCELLS+1)-1
                 X3(J,T)=.5*X1(J,T)+.5*X2(J,T)
   10 CONTINUE
      CALL PNDET(TGMASS,X3,FRAC,PND3)
*                                           ... GENERATE THETA
      THETA= .5*(-.75*PND1+PND3-.25*PND2)/(-.5*PND1+PND3-.5*PND2)
      IF(THETA.GE.1) THETA=1.0
      IF(THETA.LE..001) THETA=.001
*                                           ... GENERATE X4
      DO 20 T=1,ITMAX-1
              DO 20 J=1,EP(NCELLS+1)-1
           X4(J,T)=THETA*X2(J,T)+(1-THETA)*X1(J,T)
   20 CONTINUE
      CALL PNDET(TGMASS,X4,FRAC,PND4)
*                                          ... IF THETA IS 'GOOD' THEN STOP
*                                              OTHERWISE FOR EXTREME THETAS
*                                              NARROW INTERVAL AND CONDUCT
*                                              ONE MORE SEARCH
      IF((THETA.LT..1.OR.THETA.GT..9).AND.COUNT.LT.2) THEN
         COUNT=COUNT+1
*                                           ... CHECK TO NARROW INTERVAL
         IF(THETA.LT..5) THEN
            IF(PND4.LE.PND3) THEN
                 PND2=PND3
                 DO 40 T=1,ITMAX-1
                       DO 40 J=1,EP(NCELLS+1)-1
                          X2(J,T)=X3(J,T)
   40          CONTINUE
            ELSE
                 PND1=PND4
                 DO 50 T=1,ITMAX-1
```

```
                    DO 50 J=1,EP(NCELLS+1)-1
                       X1(J,T)=X4(J,T)
 50           CONTINUE
            END IF
         ELSE
            IF(PND4.LE.PND3) THEN
               PND1=PND3
               DO 60 T=1,ITMAX-1
                    DO 60 J=1,EP(NCELLS+1)-1
                       X1(J,T)=X3(J,T)
 60           CONTINUE
            ELSE
               PND2=PND4
               DO 70 T=1,ITMAX-1
                    DO 70 J=1,EP(NCELLS+1)-1
                       X2(J,T)=X4(J,T)
 70           CONTINUE
            END IF
         END IF
         GO TO 5
      END IF
      RETURN
      END


      SUBROUTINE SURVIV(FRAC,S)
***********************************************************************
*     PROGRAMMER: FRANK CALDWELL       DATE: SEP 87                   *
* PURPOSE:                                                            *
*   THIS PROGRAM CALCULATES THE PROBABILITY OF SURVIVING TO TIME PERIOD*
* TMAX GIVEN THAT THE TARGET IS NONDETECTED IN CELL I BY TIME T.      *
*                                                                     *
* INPUT:                                                              *
*   FRAC: A MATRIX GIVING THE FRACTION OF EACH CELL SEARCHED FOR EACH *
*         TIME PERIOD.                                                *
*                                                                     *
* OUTPUT:                                                             *
*   S: THE MATRIX OF DIMENSION NCELLS BY TMAX OF SURVIVE PROBABILITIES *
***********************************************************************
*
*                                           ... DECLARATIONS
      INTEGER TMAX,EPLEN
      PARAMETER(NCELLS=25,TMAX=10,EPLEN=26,LENGTH=200)
      INTEGER EP(EPLEN),ADJ(LENGTH),T,ADD(LENGTH),BB(TMAX)
      REAL FRAC(NCELLS,TMAX),S(NCELLS,TMAX),TRANS(LENGTH),A(LENGTH),
     1XO(NCELLS),TGTDN(NCELLS),TGTDNP(NCELLS)
      COMMON EP,ADJ,TRANS,A,XO,ADD,ITMAX,BB
*                                           ... SET S(I,TMAX) = 1 FOR ALL I.
         DO 5 I=1,NCELLS
            S(I,ITMAX)=1
  5      CONTINUE
*                                           ... ITERATIVELY CALCULATE S(I,T).
      DO 20 T=ITMAX,2,-1
*                                           ... ACCOUNT FOR SEARCH IN TIME T
         DO 10 I=1,NCELLS
            TGTDN(I)=S(I,T)*EXP(-1.0*FRAC(I,T))
 10      CONTINUE
*                                           ... TRANSITION BACKWARD IN TIME
         CALL MOVEP(TGTDN,TGTDNP)
*                                           ... SET S(I,T-1)
         DO 20 I=1,NCELLS
            S(I,T-1)=TGTDNP(I)
 20   CONTINUE
      RETURN
      END
```

LIST OF REFERENCES

1. Brown S. S., "Optimal Search for a Moving Target in Discrete Time and Space," *Operations Research*, v. 28, no. 6, pp. 1275-1289, November-December 1980.

2. Eagle J. N., "The Optimal Search for a Moving Target When the Search Path is Constrained," *Operations Research*, v. 32, no. 5, pp. 1107-1115, September-October 1984.

3. Stewart, T. J., "Search for a Moving Target When Searcher Motion is Restricted," *Computers and Operations Research*, v. 6, pp. 129-140, 1979.

4. Eagle, J. and Yee, "An Optimal Branch-and-Bound Procedure for Constrained Path, Moving Target Search Problems," a forthcoming paper.

5. Frank, M. and Wolfe, P., "An Algorithm for Quadratic Programming," *Naval Research Logistics Quarterly*, v. 3, nos. 1-2, pp. 95-110, March-June 1956.

6. Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley Publishing Company, 1985.

7. Fletcher, R., *Practical Methods of Optimization Volume I: Unconstrained Optimization*, John Wiley and Sons, 1980.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Professor James Eagle Naval Postgraduate School, Code 55er Monterey, CA 93943	1
4.	Professor Richard Rosenthal Naval Postgraduate School, Code 55rl Monterey, CA 93943	1
5.	LT James F. Caldwell Naval Submarine School, Code 20 SOAC 88020 Box 700 Groton, CT. 06349-5700	2